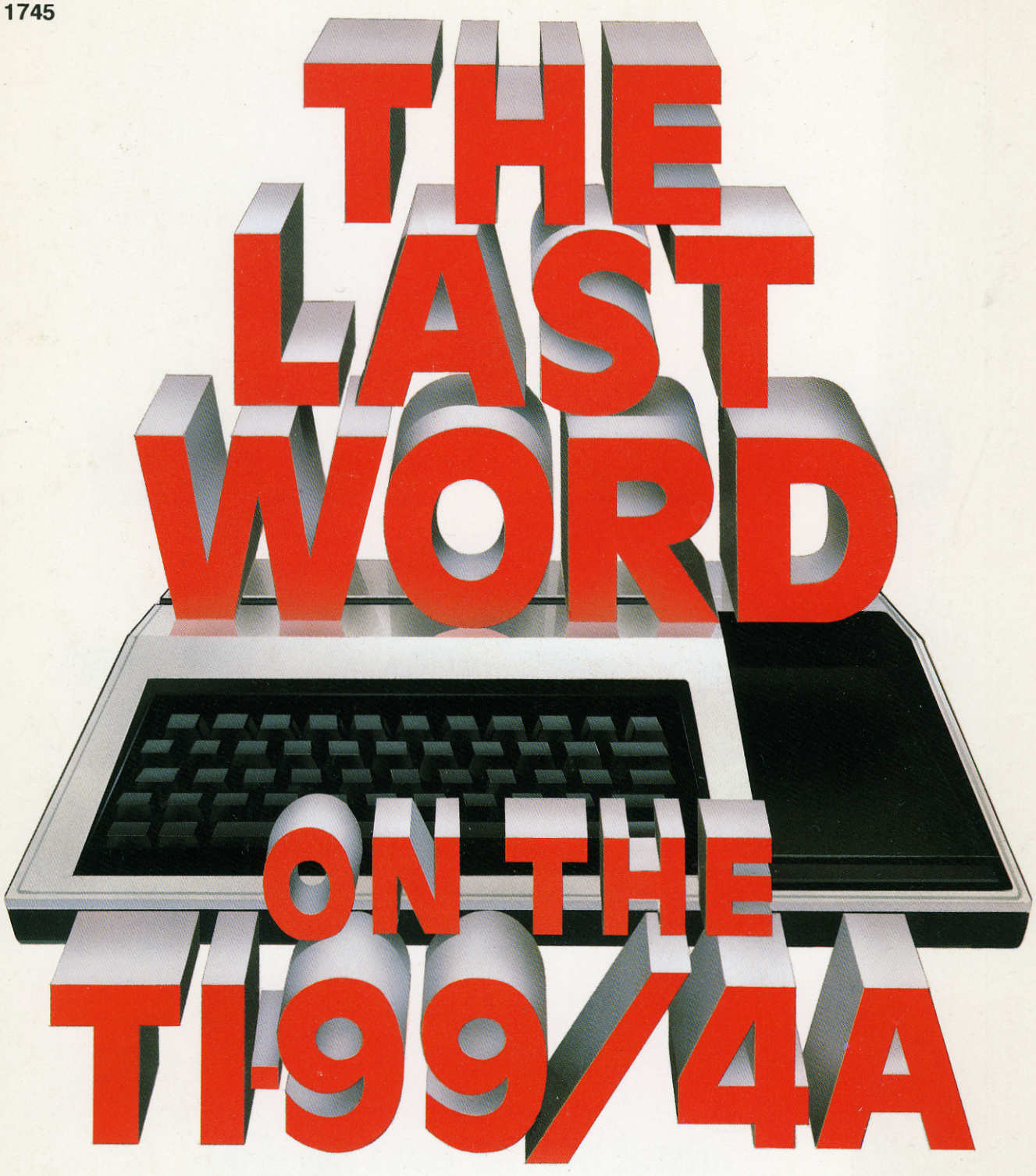
1745

THE LAST WORD ON THE TI-99/4A



BY LINDA M. & ALLEN R. SCHREIBER

THE LAST WORD ON THE TI-99/4A

BY LINDA M. & ALLEN R. SCHREIBER

TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

To Joseph Schreiber

FIRST EDITION

FIRST PRINTING

Library of Congress Cataloging in Publication Data

Schreiber, Linda M.
The last word on the TI-99/4A.
Includes index.
1. TI 99/4A (Computer)—Programming. 2. Basic (Computer program language) I. Schreiber, Allen R.
II. Title.

QZ76.8.T133S37 1984 001.64 83-24339
ISBN 0-8306-0745-5
ISBN 0-8306-1745-0 (pbk.)

Contents

Introduction

This book is designed to give you a hands-on learning experience with your TI-99/4A computer. It assumes that you have access to a TI-99/4A computer complete with the TI Extended BASIC cartridge. You do not need any previous knowledge of computers. If you can turn your system on, you are ready to begin.

The first chapters will acquaint you with your computer and the different accessories that can be attached to it. You will be introduced to new terms gradually. After you are thoroughly acquainted with your system, you will begin to program.

Each chapter introduces a few related commands. An explanation of each command is followed by an example of the way to use the command. The programs included with the chapter further illustrate the use of each new command. Each program is accompanied by a detailed explanation.

Sound, color, and graphics are included in several programs. A complete explanation of error codes is also included. You will be shown how to make your programs crash-proof, as well as how to find and correct errors in an existing program.

Once you have mastered the skills presented here, you will find that this book will serve as a handy reference guide.

Also by Linda M. Schreiber from TAB BOOKS Inc.

No. 1485 *ATARI Programming . . . with 55 Programs*
No. 1545 *Advanced Programming Techniques for Your ATARI® including Graphics and Voice Programs*

Chapter 1
What Is a Program?

Computers . . . the information age . . . a new and fascinating experience for anyone and everyone. Using computers can be fun, but exploring what can be done with them is pure delight!

Programming a computer requires only logical thinking and the spirit of adventure. You have been programmed and reprogrammed throughout your life. When your teacher gave you instructions, she was programming you. Your parents, bosses, and friends have all programmed you in some way. Think about the last time you went to the store. Did you count your change immediately after the clerk handed it to you? Why or why not? Habit—or preprogramming? You probably program your children, too!—change your clothes, brush your teeth, say your prayers, then get into bed. Carrying out a task in a logical sequence is an important element of good programming.

Computers need programs (software) to operate properly. A program is a set of instructions the computer follows. It is written in a language the computer understands. We will be writing programs in BASIC for the TI-99/4A computer throughout this book.

Programs can be very simple or very complex depending on their purpose.

```
10 REM A DEMONSTRATION OF A PROGRAM
20 PRINT "HELLO, I AM TI-99/4A COMPUTER",
30 GOTO 20
```

This is a very simple program that will display

HELLO, I AM A TI-99/4A COMPUTER

over and over again on your monitor or television screen. Your computer will continue forever if you let it. (To stop the program, press the FCTN key and the 4 key at the same time.) Your computer will also follow any instructions in the order they were given. It will not correct your spelling (unless it is programmed to), or tell you that your formula is incorrect.

The accuracy of a program depends upon the programmer. Errors occurring within a program are a result of human errors and are referred to as bugs. It is the programmer's responsibility to make the program as bug free as possible. This book will show you where bugs are most likely to appear, how to test for them, and how to correct them.

PROGRAM POSSIBILITIES

Games are a large portion of the software market. Arcade games are very popular; these programs offer the same thrills and challenges of a real arcade without the cost. The TI-99/4A can create fascinating arcade game becomes of its special graphics features. You can program your computer so you may play pinball, space games, or a shooting arcade on it.

Your computer can also be programmed for traditional family games. The graphics on your TI-99/4A allow you to easily duplicate card games, board games, or games of skill and strategy.

The remote controllers that are available for this computer help improve hand-eye coordination.

Educational

Educational computer programs are steadily becoming more popular. In the home or classroom, the computer can be a powerful and valuable instructor. It can be programmed to provide drill exercises in repetitive subjects such as math tables, states and capitals, or spelling. Your computer can also be used as a tutor for self-paced instruction.

Another effective method of learning with a computer is called simulation. A good simulation program can train a person in weeks to do what would normally take a lifetime to learn by providing the experience of both normal and extreme situations. Games can stimulate the mind as well as the imagination.

Your computer can be programmed to compose tests, store grades on a cassette or disk, average the grades, and later generate report cards.

Home Applications

Your TI-99/4A computer can be used effectively throughout your home. It can store information or help plan and organize your activities. It can be your secretary or security guard, librarian or accountant.

A program could act as a dietician, selecting menus for the week or month, and generating a shopping list for those meals. While you're at it, you may want to program the computer to recall your coupons and refunds.

Your computer is an ideal librarian. It can keep track of all your books, records, and tapes. Your program can store valuable information about anything you own.

If you have been trying to decide whether it's better to save for an item or take out a loan, write a program to show you the amount of interest your savings account would earn versus the interest that you would pay on a loan over the same period of time. Take into consideration the inflation rate and the price that the item will be by the time you have saved enough for it.

If you are buying a house, your program can show you what your mortgage payments would be over different lengths of time with varying down payments.

And, of course, you will want to program your computer to balance your checking account. Your program can also store your deductions for income tax records while it is balancing your checkbook.

In the area of health and safety, a program can help you learn first aid. With its graphics capabilities, it can teach you where the pressure points are for bleeding or how to splint a broken bone.

You can write a flower or vegetable garden program to help you plan your garden, estimate the yield of your crops, compare it to your family's needs, and show you a layout for your garden.

Darkroom enthusiasts can use a computer program to time the development of their films or store processing information.

With a special device called a modem and your telephone, you can call and connect your computer to large message centers called networks. Some of these networks serve as electronic mail boxes where you can leave a message for another person

who also belongs to this network. Other networks are giant data banks that offer UPI news, stock market reports, airline schedules, and other information.

These are some examples of the types of programs possible for your computer. Sample programs are included throughout this book. As you will become more familiar with your TI-99/4A, you will continue to discover more ideas and uses for it.

PROGRAM SOURCES

Programs for your TI-99/4A are available from a wide variety of sources. Several software firms produce well-written programs for many different applications. These programs are usually available on a cassette or disk and come with some instructions (documentation) on how to use the program. Your local computer store should be able to demonstrate these programs for you. Other programs are available only through mail order firms. Most software firms offer a catalog describing their programs and the amount of memory necessary to use them.

Another source of programs are books. Programs in books give you the opportunity to read through a program before you type it in. The best way to learn to program is by studying the programs others have written. The disadvantage of programs published in books is the time spent typing the programs into the computer. If you make a typographical error, you will have to find and correct your mistake before the program will work correctly.

Magazines are a third source of programs. There are many good articles containing programs or routines that explain the inner workings of a computer. However, unless the magazine is written specifically for the TI-99/4A, you may find that some of the programs won't work on it unless you rewrite them.

PROGRAM DIFFERENCES

Even though most popular computers on the market today are programmed in BASIC, each manufacturer chooses a slightly different dialect of BASIC. If you find a program written for another computer and the program is fairly simple, you should have no problem rewriting it for the TI-99/4A. You must also take into account the graphics the program uses. Color generation, screen resolution, and animation differ from one computer to another. Once you are familiar with your TI-99/4A and how the BASIC language works, you should be able to translate many programs written for other computers.

Most computers can also accept programs written in many other languages, such as PASCAL, LISP, PILOT, and assembly language. Each language varies from one computer to another. Each language is designed differently and has its own advantages. For most applications, the programs that you may want to write can be written very efficiently in BASIC. When timing becomes important, as with arcade games, you may want to learn assembly language.

Whether you purchase programs from a software firm or copy them from a magazine or book, you may find the program almost fits your needs. By learning to program, you will be able to change the program to suit yourself.

Chapter 2
The Makeup of a Computer

The most vital part of a computer is its central processing unit (CPU). Often no bigger than a dime, it controls and maintains the computer and many devices attached to it.

THE CENTRAL PROCESSING UNIT

The CPU can be thought of as the brain of the computer. All instructions are read and interpreted by it. It sends the correct commands to different parts of the computer and ensures that the program is followed. If you own a microwave oven, programmable video recorder, or programmable calculator, you have already worked with a CPU. The difference between the one in your computer and those in your appliances is its internal design. Your computer can be programmed for multiple uses; your microwave can only be programmed to start, stop, and cook at the correct temperature.

Every computer contains at least one CPU, often referred to as "chips." Some of the first personal computers used the INTEL 8080, others used the 6502. Both of these chips are eight-bit microprocessors. Your TI-99/4A uses a 16-bit microprocessor—the TMS 9900. This chip has some advantages over the earlier microprocessors.

The language the computer uses is called machine language. We understand it as groupings of numbers that the CPU can translate into instructions that it can follow. When we write a program in assembly language, the assembler, a software program that can read and assemble our program into a form the computer can use, translates our program into machine language. One advantage of programs written in machine language is that if a program is written for a particular CPU, it will work (within certain limitations) on all computers containing that chip. Most people don't try to program in assembly language until they have mastered BASIC.

TYPES OF MEMORY

Memory is used to store programs. Programs consist of instructions and useful information

(data). The amount of memory your computer has is measured in bytes. Some instructions use one byte of memory, others need two or more bytes. Each letter or number of data occupies one byte of memory. The longer your programs are, the more memory you need. Memory capacity is measured in thousand (K) bytes. 1K is equal to 1024 bytes, thus a 32K computer contains 32768 bytes of RAM.

There are three types of memory available to your TI-99/4A—RAM, ROM, and GROM. RAM means random-access memory. It is sometimes called read/write memory. Program data can be placed anywhere in RAM (writing to RAM) or your program can get information from any byte in RAM (reading from RAM). RAM should never be used for permanent storage since it can't retain information once the power is shut off. It is needed for programming because it can be easily changed by the user or under program control.

Static RAM is used in some computers. This type of RAM is stable. Once an instruction is placed into it, it will retain the instruction until it is changed or the power is shut off.

Dynamic RAM is used in most popular home computers, including the TI-99/4A. After an instruction is placed in this type of memory, the CPU must constantly refresh (remind) the memory of the information placed there. This makes the CPU run slower than it would with static memory, but for most applications, this is not crucial. Some devices cannot run properly with dynamic memory, but they are few. Dynamic RAM is much less expensive than static RAM.

Memory from different manufacturers may have different access times, that is, the amount of time the CPU has to retrieve an instruction from memory. Speed becomes an important factor if your program must do many different calculations before arriving at the answer. In most programs, though, you will not notice the difference in speed.

ROM means read-only memory. Your TI-99/4A has its operating system in ROM. The program on these ROM chips has been permanently fixed and will remain there whether the unit is turned on or off. When you turn your TI-99/4A on, this program begins immediately. It checks to see if you have inserted a cartridge and taken care of many other tasks before the screen appears on your monitor.

The cartridges that contain Extended BASIC, or one of the many games available for the TI-99/4A contain GROM chips.

You can add more memory to your TI-99/4A with the expansion interface. There is a slot in this interface for 32K RAM. You can also use this interface for the disk drive and other devices. The mini memory module is a unique cartridge. You can store a program on it and remove it from the computer; the program will remain in the cartridge. There is a small battery in the unit that keeps the memory active. You can erase the program with a special command and store a new program on the cartridge.

MASS STORAGE

Once you have written a program, you will want to save it for future use. Cassette recorders are an inexpensive and easy-to-use way to store programs. Once a program is placed on the cassette it will stay there until it is erased or recorded over. The computer records the program on the tape by generating two tones. These tones represent the instructions in the program. This can be done because the most basic instructions used by the computer, called machine code, is a binary code consisting of combinations of ones and zeros. The computer loads a program from the cassette by listening to the tones and translating them into the corresponding binary digits.

Cassettes are inexpensive, easy to use, and, because they are in a plastic case, easily handled by children. They can be shipped or stored with minimum precautions. Programs save and load to cassettes very slowly, and you cannot access the information on them easily. If you purchase inferior tapes, ones that are too thin, you run the risk of having your program destroyed by the recorder. If you do not wish to purchase tape designed for computer use, you may use recording tape, but don't use long-playing ones (45 to 120 minutes). This tape is too thin and easily damaged by the recorder.

A more efficient way of storing programs is with a disk drive. There are two different drives available for the TI-99/4A—the single-sided drive and the double-sided drive. The double-sided drive can store program information on both sides of the disk. This doubles the amount of information that the disk can hold.

A floppy disk (sometimes called a diskette) is a thin Mylar circular medium similar to a record. It is covered with a thin jacket. There is a slot cut out on both sides of the jacket exposing the surface of the disk (Fig. 2-1). Touching this surface could damage the disk. Programs are stored on disks by electronic impulses that magnetize the surface. Because the disk spins rapidly inside the drive, the computer can save or load a program on it much faster than on a cassette. Also, the disk has tracks much like a record has grooves. Any part of the disk can be accessed at any time.

Disks are very vulnerable to static charges. An electrical charge, even a mild one produced by walking across the carpet, can destroy the programs on the disk. The jacket on the disk is for protection against dust and dirt. If the disk is bent, it will not spin properly, and the computer will not be able to read the program or data stored on it.

Since both disks and cassettes store information magnetically, you should not place them near a magnetic field such as the top of a speaker, a motor, or a monitor (television). A strong magnetic field could destroy your program.

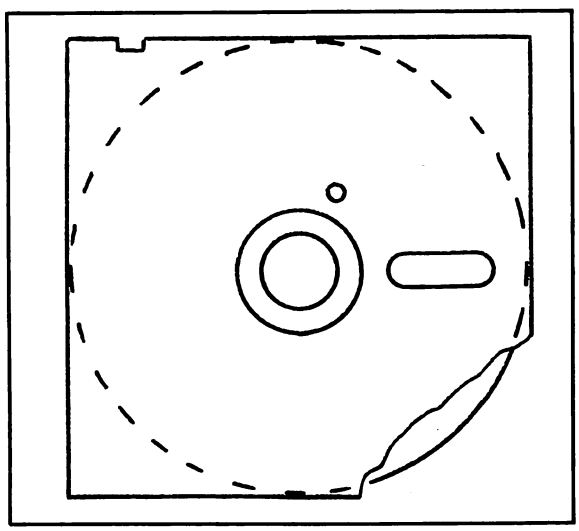


Fig. 2-1. Cutaway of a disk.

ACCESSORIES

Many other accessories, or peripherals, are available for and compatible with the TI-99/4A. These peripherals can be connected to your TI-99/4A through the expansion interface or through the port on the right side of the computer.

The expansion interface is a large metal case that houses extra memory, the disk drive, and the disk controller card. There are several other slots available in the expansion interface for other peripherals.

By adding the TI Printer to your computer, you can get a listing of your program on paper. When you write long programs, you may find it difficult to remember different parts of the program. By getting a listing or *hard copy* of your program you are able to read the entire program, compare different sections of the program, or check the program more easily than when you are reading it from the screen.

The TI Acoustic Coupler Modem can connect your TI-99/4A to the outside world. The word modem means modulator-demodulator. That is, it can change the signals that you send from your computer into signals that can be transmitted over the telephone lines. Demodulator means it changes the signals that it receives from another computer over the telephone line into signals that your computer can understand. There are several network services available that can provide you with up-to-the-minute stock market reports, UPI transmissions, or software for your computer. The modem allows you to connect with message centers that serve as electronic mailboxes in certain areas of the country. Often a computer club will host such a message center.

Your computer can also speak through TI's Solid State Speech Synthesizer. Many educational programs as well as games use this feature.

Your TI-99/4A computer can be connected to your color television or a color monitor. A monitor is essentially a television without a tuner. A color monitor will provide you with a clearer, crisper

picture than a television set, but for most applications, a television will do fine.

Using the keyboard to play arcade games just doesn't feel quite right. These games need a quick response from the player who shouldn't be fumbling with a keyboard. Along the left side of your TI-99/4A is an outlet or port. By using a "Y" shaped connector, you can add two joysticks. The joystick is a rectangular box with one stick and a button on it. This stick sends signals to the computer. Your program can determine if you are moving the stick or pressing the button.

We will be using some of these accessories later in this book.

Chapter 3
Getting to Know the Keyboard

The best way to learn about your TI-99/4A computer is to use it. Remove any cartridge that you might have in the computer, turn on your television or monitor, and turn on your computer. Your screen should display two bright color bands near the top and bottom. The TI logo and name should appear in the center of the screen, and

READY-PRESS ANY KEY TO BEGIN

should appear just above the bottom color band. Press any key.

The screen will change. A short menu will appear. A menu is a listing of choices that you can make. Since there is no cartridge in the computer, your only choice is number 1 for TI BASIC. Press the key with the number 1 on it.

The screen clears for a few seconds. The words TI BASIC READY are displayed near the bottom. Under these words a caret (>) and a dark flashing square appear. This square is called a cursor. It marks the position that the next character will occupy.

STANDARD CHARACTERS

Look at your keyboard. Most of the letters and numbers are in the same place as a standard typewriter keyboard. Some of the characters may be in a different place than your typewriter.

Your TI-99/4A has two sets of letters in its memory. One set is the standard uppercase letters. The second set is a smaller version of the uppercase letters. The TI-99/4A does not display true lowercase letters with the letters in its memory.

The right most key of the center row is called ENTER. This key acts as a carriage return. It moves the cursor down one line and to the left side of the screen.

SPECIAL FUNCTION KEYS

Along the top ledge of your computer is a strip with different functions on it. Look at the key to the

right of the space bar. This is the function key. On the front side of this key is a grey dot. On the left side of the space bar is the control key. This key has a *red* dot on the front side of it. Look again at the strip on the top ledge. On the right are both the red dot and the grey dot. The functions listed on the same line as the grey dot operate when the function key and the number key under the command are pressed at the same time. The functions listed on the same line as the red dot operate when the control key and the number key under that command are pressed at the same time.

Hold down either the function key or the control key. Now press the key with the plus sign (+) on it. This is the QUIT command. Your screen should clear and the title page with the color bars should appear on the screen. Press any key, then the number 1 to return to TI BASIC.

The key to the left of the control key is the ALPHA LOCK key. This is similar to a shift-lock key. By pressing the ALPHA LOCK key into the down position, you lock the computer into using only capital letters. Press the ALPHA LOCK key so that it is in the raised position. This unlocks the keyboard, letting you type in both large uppercase and small uppercase letters. The number and symbol keys differ from those on a typewriter keyboard; you must press the shift key for the symbols above the numbers, the greater than or less than symbols, or other character keys even if the ALPHA LOCK key is in the down position. Most programmers use capital letters exclusively (BASIC doesn't recognize lowercase commands) so the ALPHA LOCK key is normally depressed. It would be cumbersome to have to unlock the ALPHA LOCK key for the numbers.

The function key (FCTN) is also used with certain other keys. Look at the letters: W, E, R, T, U, I, O, P, A, S, D, F, G, Z, X, C. Each of these keys have another character printed on the front of it. You cannot use the shift key to print these characters. Instead you press the FCTN key and the character key at the same time. For example, every time you want a quotation mark, you hold down the FCTN key and press the P key.

Holding down FCTN and pressing the 4 key can serve one of two purposes. If you are running a BASIC program and you want to stop the program, hold down FCTN and 4. The program will stop and the screen statement will tell you what line the program stopped at. If you are typing in a program and you decide that the line has several errors on it, or wrong commands, instead of using FCTN and the backarrow to delete the line, you can hold down FCTN and press the 4 key. The computer will ignore the line and bring the cursor down one line. You can now enter a new line.

Holding down FCTN and the 3 key is very similar to using the 4 key. When you press the 3 key with FCTN to erase, the cursor moves to the beginning of the line, erasing everything that you typed. The screen does not scroll. The cursor remains on the same line.

FCTN and the 2 key are used to insert more characters into a line. Type this short exercise:

```
30  PINT "THIS IS A TEST"
```

do not press the ENTER key. Notice that the word PRINT is misspelled. Press the FCTN and the S key. The cursor moves back one position. Hold down both keys until the cursor is over the "I" in "PINT." Now hold down the FCTN and press the 2 key. Release both keys. Press the R key. The "R" will be inserted between the "P" and the "I." If you had more letters or characters to insert, you could keep typing, and each character would move the letters and characters in the line to the right one position. If the rest of the line is correct, you can press the ENTER key. If you want to enter more commands at the end of the line, use FCTN and D to move the cursor to the right.

To delete characters from a line, use FCTN and the 1 key. Type this example:

```
30  PRINT "THISS IS A TEST"
```

again, do not press the ENTER key. Instead, hold down FCTN and press S. Move the cursor to the left until it is over one of the "Ss" in THIS. Hold

down FCTN and the 1 key. The "S" on the screen will be removed. As long as you hold down both FCTN and 1 characters will be removed from the line.

These keys are very useful when entering programs. If you make a mistake while you are entering a line, you can easily move the cursor and correct the error.

ACCESSORY OUTLETS

On the left side of the computer is an outlet. This outlet is the connector for your remote controllers. You can use any standard joystick with your TI-99/4A. Using a connector that is shaped like a "Y," you can adapt your TI-99/4A to use two joysticks.

On the right side of the keyboard is a large connector you can connect your interface to. If you own a speech synthesizer, it will also connect on this side. The speech synthesizer is designed to fit between the computer and the interface. Some disk drives that are not contained in the expansion module also connect here.

On the top surface of the computer, to the right of the keys, is a large, flat area. The GROM cartridges are inserted here. These are the cartridges that TI and a few other software firms manufacture. If you own the TI Extended BASIC cartridge, you will insert it here.

On the back of the computer, near the side of the interface connector is a small rectangular connector. The cables for a cassette recorder are connected here. This special set of cables should be available through the same store that sold you your TI-99/4A. If you do not have a disk drive, then you will need these cables so that you can save your programs to a cassette.

Near the other corner on the back of the computer is a circular connector. This is where the cable for the video monitor connects. The TI-99/4A video monitor comes with its own cable and uses this connector.

Chapter 4
Organizing Your Program

No matter how creative a program appears, the rudiments of programming are the same. Very few programmers can conceive an idea, sit down at the keyboard, and enter the program without a plan or guideline. Good programs are carefully thought-out and developed. Consideration is given to the parts of the program that the computer will perform, the information the user will provide, and the information stored in the program.

Let's say you would like to write a program that will determine the cost of the floor covering for a room with a complete cost comparison of the different floor treatments possible. This program would consist of several small programs, or routines. This chapter develops a portion of that program. The program computes the area of the floor in square feet and square yards.

PARTS OF A PROGRAM

The computer will calculate the area of the floor, the amount of floor covering needed, the price of the floor covering, and the cost per year, determined by the average life of the floor covering. To do this, the computer must be given the essential facts, including the measurements of the room, the price of the floor covering, and what flooring is being considered. This information is provided by you, the user. The computer also needs information about the expected life of floor coverings, the conversion from square feet to square yards, and the pricing formula. All these figures remain constant and can be stored in the program.

The set of instructions the computer will follow regardless of the information entered is called the algorithm. The answers to the questions that the program asks are supplied by the user and will change from person to person depending on the questions and the circumstances. Errors (or bugs) can be generated if the user enters incorrect information. The information stored in a program and used to perform calculations is the data base. If the data is incorrect, the outcome of the program will also be in error.

FLOWCHARTS

A flowchart is an outline of a program that the programmer uses to develop the program. It serves as a guide, showing the parts of the program that must be included for it to function correctly. To program without a flowchart would be like trying to take a trip to an unknown region without a map. It can be done, but it can also be a waste of time and energy.

Every programmer develops a personal style of flowcharting. There are several well-known types, including Warner-Orr diagramming, data-flow diagrams, structure charts, and structured pseudocode. Throughout this book the standard symbols, shown in Fig. 4-1, are used.

The *terminal* symbol is used to indicate the beginning and ending of the program. *Input/output* indicates where the user must provide information, the program will *read* its own data base, or information will be printed to the screen or printer. The decision symbol indicates where the computer will have to determine which set of instructions to follow. *Predefined process* is the sequence of program statements (instructions) the computer will follow regardless of what has been entered by the user. The *connector* is used to show that the flowchart continues on another part of the page, or even to another page. The connecting connectors will have the same number inside the circle.

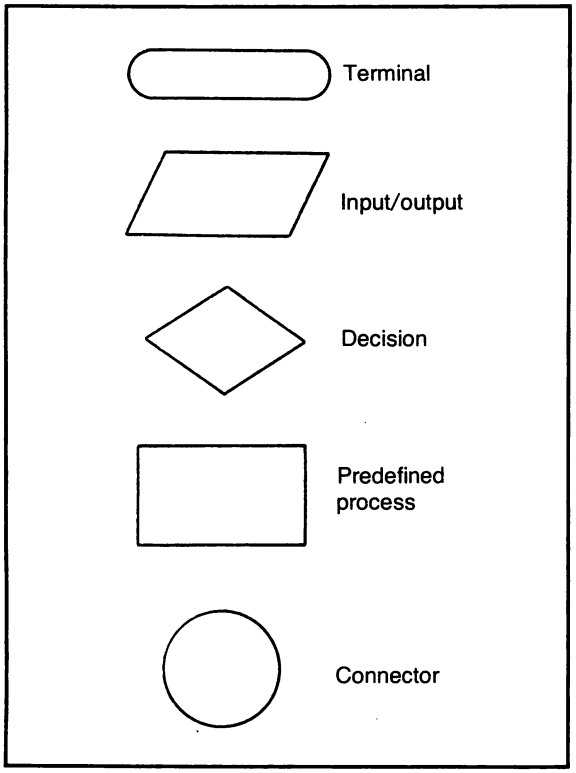


Fig. 4-1. Standard flowchart symbols.

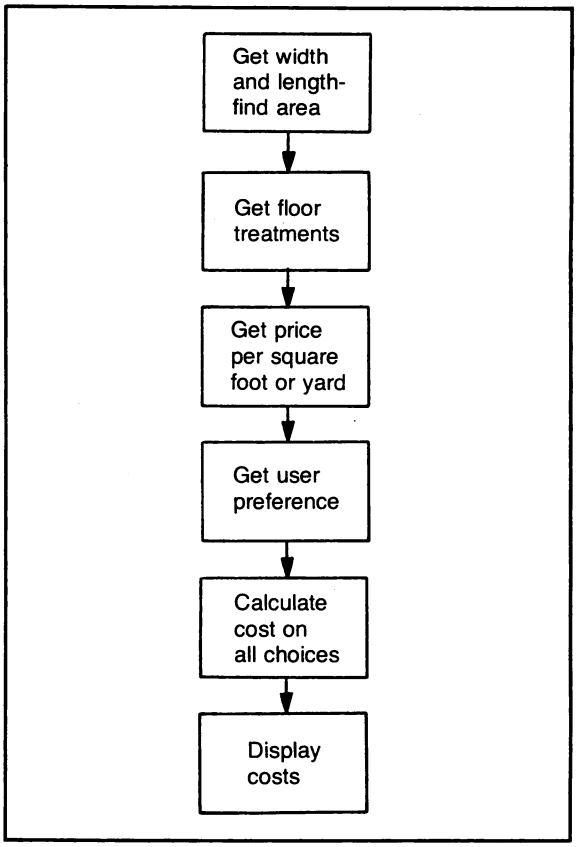


Fig. 4-2. Flowchart indicating main routines of a program.

When you flowchart a large program, you may find it helpful to divide the program into several small modules before you draw a detailed flowchart.

Figure 4-2 is a block diagram of the different parts of the program. The first block indicates the *routine*, or module of instructions, that determines the size of the room. The next three modules determine the different treatments being considered,

the price (in cost per yard), and the user's preference. The program computes the cost of the treatment in terms of the overall price and the price per year over the expected life. The program would show the user the most expensive treatment, the least expensive treatment, and the cost of the treatment that the user prefers. A good program would give the user the option of changing some of the treatments or adding new ones. The end result would be the amount of material needed to cover the floor and the approximate cost. Each of these modules can be flowcharted with a very detailed flowchart. Figure 4-3 is a flowchart containing the routine for the first module of the program.

PUTTING THE PROGRAM ON PAPER

Jot down your program idea after you've thought it out, using the block diagram. Now think . . . what is the best way to handle the details of the program? Look again at Fig. 4-3. The first thing the program does is ask the user for the dimensions of the room. The program needs this information. Request it first, not after you ask whether the user will tile or carpet the floor. Any facts that are vital to the program should be asked for as soon as possible. The message written on the side of the flowchart is a remark, a reminder to the programmer why this command should be included in the program, or an explanation of how this part of the program should work. The more remarks you make, the clearer your program will be.

The next part of the flowchart requests the type of flooring and the cost per square foot. The *diamond* reading ANY MORE? indicates a decision the computer will make. If the user say that there are more types of floorings to be entered, the program will go back to the step asking for the type of flooring. If there are no more entries, the program will continue.

The computer determines whether more than one entry was made. If so, it requests the user's preference, then computes the cost and cost per year. The last part of the program shows the user the costs of the preferred treatment, the most expensive treatment, and the least expensive treatment. It also indicates the best floor treatment based on the average cost per year.

The size of the room, and the types of flooring and their costs are data the user inputs. The squares in the flowchart are the algorithms or instructions that the computer will follow to reach an answer. The data base is not easily discovered by reading the flowchart. When the program computes the cost per year, it will use the figures stored in its data base. This information must be accurate if the program is to be accurate.

The program starts at the top of the flowchart and works its way to the bottom; It rarely backtracks. This is good programming practice; if your program jumps from one routine to another, you will become confused writing it, and if a bug should appear, it will take much longer to correct it. Divide your program into small routines, so you can write cleaner programs with less chance of errors.

Listing 4-1 is the BASIC listing of the first flowchart routine. This program will work in TI BASIC or TI Extended BASIC. The remarks in the program correspond with the instructions in the flowchart. Below is a line by line explanation of the program:

Listing 4-1

Lines 100-130 are remark lines. They name the program and give general information about it.

Line 140 contains the command to clear the screen. This command removes everything that is presently on the screen. Keep your program presentation neat—clear the screen to get rid of old information.

Line 150 prints a question on the screen. The program would like the length of the room in feet.

Line 170 waits for the user to enter the length. The amount entered will be stored in the LENGTH variable.

Line 180 prints the next question. Now the program would like the width of the room in feet.

Line 200 waits until the width is entered. The program stores this number in the WIDTH variable. (The numbers entered in lines 170 and 200 will change each time the program is used.)

Line 220 computes the are in square feet.

Line 240 changes the square feet into square yards. (These two algorithms, Lines 220 and 240, re-

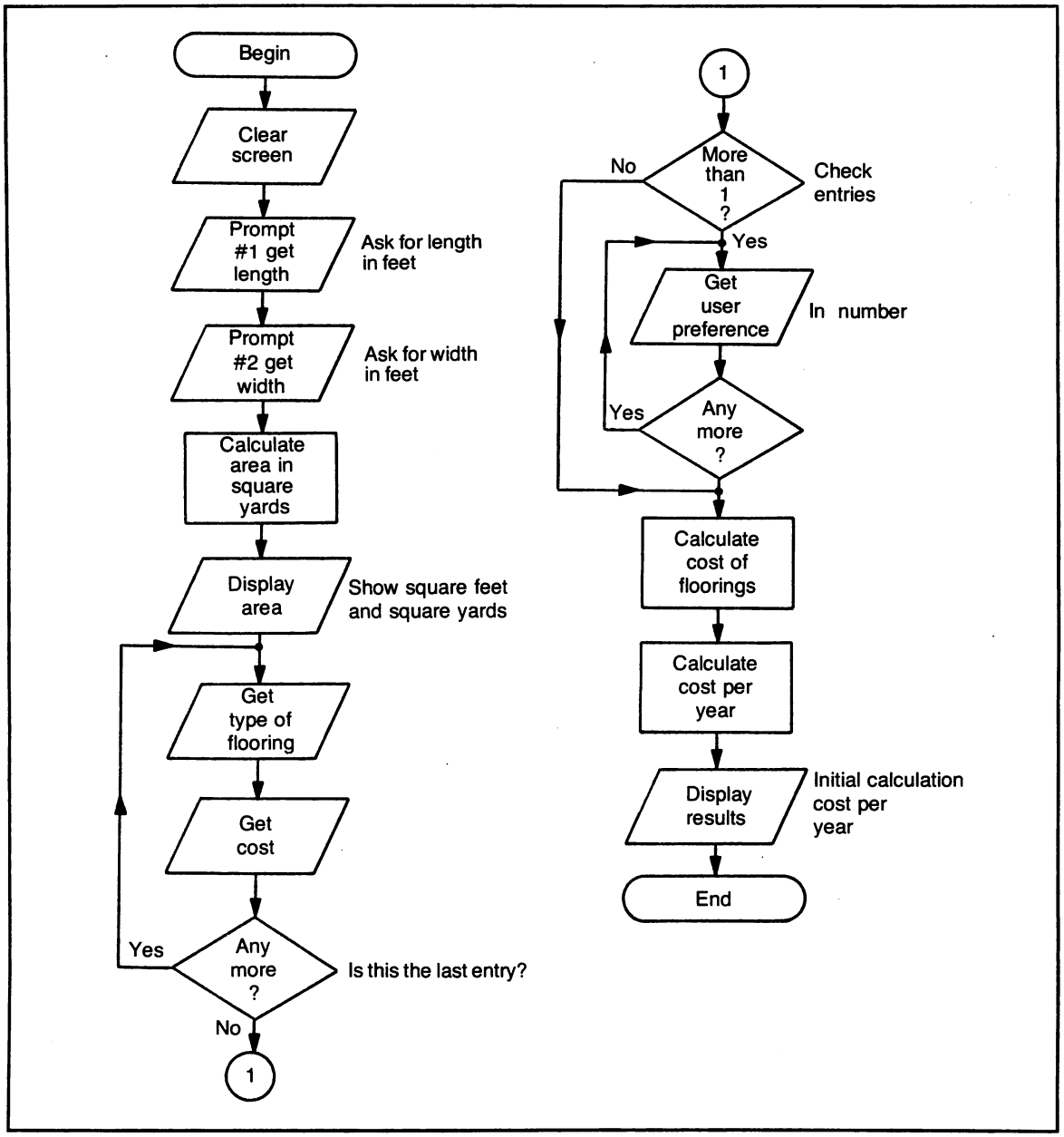


Fig. 4-3. Detailed flowchart for Listing 4-1.

main the same no matter what size the room is.)
Line 260 rounds the square yards to the nearest square yard.
Line 270 prints a message on the screen.
Line 280 prints the area of the room in square feet.
Line 290 prints the area of the room in square yards.
Line 300 tells the computer that the program has ended.

Listing 4-1

```
100 REM LISTING 4-1
110 REM COMPUTE SQUARE FEET AND SQUARE Y
ARDS
120 REM L.M.SCHREIBER FOR TAB BOOKS
130 REM CLEAR THE SCREEN
140 CALL CLEAR
150 PRINT "What is the lensth of the   r
oom (in feet)";
160 REM STORE LENGTH IN THE VARIABLE 'LE
NGTH'
170 INPUT LENGTH
180 PRINT : : :"What is the width of the
room (in feet)";
190 REM STORE THE WIDTH IN THE VARIABLE
'WIDTH'
200 INPUT WIDTH
210 REM COMPUTE THE SQUARE FEET
220 AREA=WIDTH*LENGTH
230 REM COMPUTE THE SQUARE YARDS TOO
240 SQYARD=AREA/9
250 REM ROUND OFF TO NEARESTSQUARE YARD
260 SQYARD=INT(SQYARD+0.5)
270 PRINT : : : :"The area of the room i
s :"
280 PRINT :AREA;"square feet"
290 PRINT SQYARD;"square yards"
300 END
```

Chapter 5
Commands, Statements, and Functions

There are two ways that you can communicate with your TI-99/4A. You can type an instruction; then press the ENTER key and the computer will immediately execute it, or you can enter a series of commands in a program to be executed in sequence. In the first example, the instruction that you give the computer is a direct command. In the second example, the lines of a program contain the instructions, which are indirect commands or program statements. In the following programs we will use the commands for TI Extended BASIC. When the command is used differently by TI BASIC, we will show examples for both versions of BASIC.

DIRECT COMMANDS

Most commands can be used as direct commands. Many direct commands can also be used in a program. When you type RUN to start a program, you are giving the computer a direct command. An entire line of a program can be entered as a direct command. If you are using the Extended BASIC cartridge, try this:

Type: FOR X=1 TO 10::PRINT X::NEXT X
Press the ENTER key.

The left side of your screen should display the numbers from 1 to 10 along the left side. You could also type:

```
10  FOR X=1 TO 10::PRINT X::NEXT X (ENTER)
    or in TI BASIC type:
10  FOR X=1 TO 10
20  PRINT X
30  NEXT X
40  END
```

For either BASIC type: RUN
The results should be the same.

NEW

One direct command that should be used sparingly is NEW. This command erases the program that is in the computer's memory. It cannot be used as a command in a program. It is best used

when and only when you have finished a program, saved it, and want to enter another program. If you get into the habit of using this command without much thought, you may find that you have just wiped out two or more hours of hard work.

The NEW command does have its advantages, however. Since it clears a program out of memory, you can begin typing another program and not have lines left over from the last program. You do not have to enter the NEW command before loading a program from cassette or disk because this is done automatically by the computer.

BYE

Another direct command is BYE. Type this when you want to return to the TI logo page. It is the equivalent of pressing FCTN and the + key. Your entire program will be erased from the computer's memory.

PROGRAM STATEMENTS

The instructions in the numbered lines of a program are program statements. They are entered when you type in a program or load a program from the cassette or the disk. The computer stores these statements in its RAM. It will follow these instructions when the program is RUN.

Each program statement must begin with a line number. Most programmers start with 10 and number the lines in multiples of ten. This lets you easily add lines to your program without reorganizing or retyping an entire routine.

NUMBER

One feature of TI BASIC and TI Extended BASIC is the NUMBER command. This command allows you to enter the lines of a program without having to enter the line numbers. Your program lines will be numbered automatically. There are four different ways that you can use this command. The abbreviation or shorter version of the command is NUM. Using the NUMBER command by itself tells the computer to begin with line number 100 and add 10 for every additional line. Your screen might look like this:

```
NUMBER
100  REM PROGRAM FOR AREA
110  PRINT "ENTER THE LENGTH"
120  INPUT L
```

By entering two numbers after the NUMBER command, you can tell the computer which line number to start with and how far apart the line numbers should be as illustrated below.

```
NUMBER 500,5
500  REM PROGRAM FOR THE AREA
505  PRINT "ENTER THE LENGTH"
510  INPUT L
```

If you do not enter the last number, the computer will produce line numbers in multiples of 10, like this:

```
NUMBER 300
300  REM PROGRAM FOR THE AREA
310  PRINT "ENTER THE LENGTH"
320  INPUT L
```

If you do not specify which line to start at, the computer will begin with line 100.

```
NUMBER ,20
100  REM PROGRAM FOR THE AREA
120  PRINT "ENTER THE LENGTH"
140  INPUT L
```

The numbers that the computer uses when you do not enter a specific number is called default value. The default values for NUMBER are 100 (first line number) and 10 (apart).

RESEQUENCE

The NUMBER command keeps your program well organized while you are writing it. Remember, we want to keep the line numbers in multiples of 10 so that we can add lines to our program if we need to. Looking at the first example, let's add one line to the program.

```
105  PRINT "THE AREA IS THE PRODUCT OF
       THE LENGTH TIMES THE WIDTH"
```

The computer will automatically add this line between lines 100 and 110. By using the RESEQUENCE command, we can renumber the program, keeping all the lines evenly spaced.

Type: RESEQUENCE

Now LIST the program. It should look like this:

```
100  REM PROGRAM FOR AREA
110  PRINT "THE AREA IS THE PRODUCT OF
     THE LENGTH TIMES THE WIDTH"
120  PRINT "ENTER THE LENGTH"
130  INPUT L
```

The RESEQUENCE command is very similar to the NUMBER command. The default values are 100 for the beginning line number, and 10 for the lines between line numbers.

Try these commands with the program:

RESEQUENCE 500,20

The program will begin at line 500 and the lines are 20 apart.

RESEQUENCE 300

The program will begin at line 300 and the lines are 10 apart.

RESEQUENCE ,20

The program will begin at line 100 and the lines are 20 apart.

In TI Extended BASIC, program statements can not exceed five screen lines in length. However, long lines often mean multiple statements on each line. Too many commands on one line is not a good programming practice. Lengthy lines can confuse the programmer and are sometimes impossible to debug.

There are times, however, when you will need to put two program statements on the same line, for example, when you want the program to make a decision. Place two (2) colons between the end of one statement and the beginning of the next. The colons tell the computer not to go on to the next line but to look at the rest of this line. The line you typed at the beginning of this chapter is an example of a line with multiple statements.

REM

There is one program statement that the computer will always ignore, even though it is most useful to the programmer. That is the REM (remark) statement. Use the REM as a reminder to yourself about what the routine does, why you did it, when it will be used, and the like. Often a good routine is extremely confusing without remark statements if you haven't looked at it for a long period of time.

In TI BASIC the REM statement must be used on a program line by itself, as shown here:

```
20  REM CLEAR THE SCREEN
```

In Extended BASIC, the exclamation mark (!) can be used to replace the word REM as illustrated below.

```
20  ! PROGRAM ON VARIABLES
30  A=10 ! SET THE FIRST VARIABLE
```

The double colons are not necessary to separate the program statement from the remark.

END

When the computer comes to a line that has the END statement on it, it will stop running the program. The END statement is not necessary if the program starts with the first line and continues through to the last line. Sometimes, however, a program is written with routines or program sections at the end of the program, and the main portion of the program occupies the first hundred or so lines. Placing the END statement between the main program and the routines will tell the computer that the program is over and that it should not execute the program lines following the END statement. The two listings that follow will illustrate the difference between a program with subroutines and a program without them.

```
10  REM A PROGRAM WITH NO SUBROU-
    TINES
20  PRINT "PLEASE ENTER A NUMBER"
30  INPUT A
40  PRINT "PLEASE ENTER A NUMBER"
50  INPUT B
60  PRINT "THE PRODUCT OF ";A;" and
    ";B;" IS ";A*B

10  REM A PROGRAM WITH A SUBROUTINE
20  GOSUB 100
30  A=C
40  GOSUB 100
50  B=C
60  PRINT "THE PRODUCT OF ";A;" AND
    ";B;" IS "A*B
70  END
100 PRINT "PLEASE ENTER A NUMBER"
110 INPUT C
120 RETURN
```

STOP

The STOP command is similar to the END command. When the computer comes to the STOP command in a program line, it stops running the program.

VERSION

The VERSION command can be used to find out which version of TI Extended BASIC is being used in the TI computer. There are different versions of the Extended BASIC cartridge. The changes in the BASIC are not apparent to the user, but for someone who is developing programs for commercial uses, and is using parts of the operating system that could be different in the old or newer versions of TI Extended BASIC, this command is very useful.

```
10  CALL VERSION(T)::PRINT T
```

The version of TI Extended BASIC, used for this book is 110.

EDITING

Editing program lines is something that all programmers learn to do sooner or later (usually sooner). You'll use the editing feature to correct typing errors; to change values; to fix errors in the program, commands, or operation, and to delete unnecessary instructions or to add instructions. The TI-99/4A has some very good editing features. There are differences between the editing features of TI BASIC and TI Extended BASIC. For instance, if you enter a statement incorrectly, the computer will not understand it and stop. An error message will be displayed on the screen. To make the program run correctly, you will have to correct the program statement. On the other hand, if you tell the computer to print a word that is misspelled, the program will do so. The computer cannot tell if a word is wrong unless it is a command word.

To edit a program line in TI BASIC, type:

```
EDIT (line number)
```

For example, your line 20 reads:

```
20  PINT "THIS IS A TEST"
```

There is an "R" missing from the word PRINT. After you type EDIT 20, line 20 will reappear on the screen. The cursor will flash over the first letter of the program statement. Now you can use the FCTN and right arrow key to move the cursor over the "I". Use the FCTN and 2 key to enter the insert mode. Press the letter R and it will be inserted between the "P" and the "I." You can now press the ENTER key and the program statement will be corrected.

Once you have entered the EDIT mode by typing EDIT and the line number, you can use FCTN along with the right or left arrow keys to move the cursor on the program line. Use FCTN with the 1 or 2 key to delete or insert characters.

In TI Extended BASIC, there is no EDIT command perse. To edit the program statements, simply enter the line number, and press FCTN along with the up arrow (W) or down arrow (X) key. Again, that entire line will appear on the screen with the cursor flashing over the first letter or character in that program statement.

To add letters or characters to the program

statement, use the function key and the right or left arrow key to move the cursor. Let's use the same program line as before.

```
20 PINT "THIS IS A TEST"
```

Type 20 and while pressing FCTN, press the up arrow (W) or down arrow (X) key. Line 20 will reappear on the screen with the cursor flashing over the "P." Press FCTN and the right arrow (D) key. The cursor will move over the "I." This is where the "R" should be. Now press FCTN and the 2 key. Press the R key. The "R" will appear between the "P" and the "I." Press ENTER and the line will be corrected.

To delete letters or characters from a program line, use the same procedure, except, instead of pressing the 2 key, press the 1 key. The letters or characters under the cursor will disappear. Press ENTER when all the characters that you want to erase are removed from the program line.

To change a line completely, you can enter the program line and begin typing the new line. However, if you are already in the EDIT mode because you entered the line number and FCTN/up arrow, you can press FCTN and the 3 key. The entire line will be erased from the screen. You can now begin typing the new line.

Changing Line Numbers

Sometimes you may want to use the same or very similar program line several times in your program. You do not have to type the same line over and over again with new line numbers. There are two different ways to change line numbers in TI Extended BASIC.

Enter the following program statement:

```
20 PRINT "THIS IS FUN"
```

Now press FCTN and the 8 key. The entire line is reprinted on the screen with the cursor flashing over the line number 2. Type the number "3". The number "2" is replaced with the "3." Now use FCTN with the right arrow (D) key to move the cursor over the "F" in "FUN." Press FCTN and 2 for the insert mode. Now enter the words "MUCH MORE" and press ENTER. Type LIST to see the entire program. Your screen should display:

```
20 PRINT "THIS IS FUN"
30 PRINT "THIS IS MUCH MORE FUN"
```

You entered two program lines without having to retype the second program line.

(Pressing FCTN with 8 will always display the last command or line that was entered. If you pressed those keys now, the LIST command would be displayed on the screen.)

When you are entering a program, you don't always know that you will want to reuse a line in other parts of the program, or that you may want to move a program line to another part of the program. To renumber a program line that was *not* just entered, type the program line. Press FCTN and the up arrow (W) or down arrow (X). Now using the FCTN and the right arrow, move the cursor to the space after the last letter or character in the program line entered. Press FCTN with 8 and this line will reappear on the screen with the cursor flashing over the program line number. You can now change the program line number.

When you use FCTN with the 8 key to change line numbers, you reenter the line with a new number. The original line remains in the program.

Deleting Lines

To remove or delete a program line, type the line number and press ENTER. If you are in the edit mode, you can use FCTN and the 3 key to erase the line, then press ENTER. The program line will be removed from the program.

ERROR MESSAGES

Your TI-99/4A may tell you that your program contains an error when you are entering a program line, after you type RUN but before the program is actually executed, or while the computer is running your program. Usually, the error message will also contain the line number of the error. Although most

messages are self-explanatory, the following guide offers suggestions on how to avoid or correct the most common errors. They may not help you just yet, but it is a good idea to be familiar with them.

ERROR	CAUSE and CORRECTION
MEMORY FULL	The program uses more memory than available in your computer. Check the dimension statement. You may be setting aside more memory than you need to. Divide your program into smaller programs that can be chained together. Look at your program lines that contain the GOSUB command. Make sure that you are not calling the same line that the command is on. Check for a RETURN at the end of every subroutine.
BAD ARGUMENT	You may be trying to find the ASCII value or numeric value of an empty string. If the string does contain information, be sure that you are taking the value of a number, not letters.
NAME CONFLICT	A name used for a variable cannot also be used for an array or function. Arrays, variables, and functions cannot duplicate each others names.
BAD SUBSCRIPT	You are trying to use a subscript in an array that is greater than the limits of the array, or you are using the subscript 0 and the base 1 option was chosen. Subscripts must also be integers.
DATA ERROR	The problem may lie in the DATA lines or the READ command. Be sure that there are commas between the elements in the DATA line. Either there is not enough data in the DATA lines, you want to access the same data but did not use the RESTORE command, or you are trying to RESTORE a line number that is higher than the last line of the program.
INPUT ERROR	The program needs a number but a letter or character was entered. If a letter or character is supposed to be entered, change the variable to a string variable.
BAD LINE NUMBER	A GOSUB, GOTO, or THEN command referred to a line not in the program. Correct the line number in the program line, or add the missing line to the program.
FOR-NEXT ERROR	The number of program lines that contain the FOR statement do not match the number of program lines that contain the NEXT statement. Check nested loops for a missing NEXT or too many NEXT statements.
LINE TOO LONG	The line is too long for BASIC to understand. Shorten the line. If it is a DATA line, count the number of elements in that line. Only 30 commas are allowed on a line of data.
CAN'T DO THAT	A RETURN command cannot find the matching GOSUB. If you place your subroutines at the end of your program, be sure that there is an END statement before the first subroutine. Make sure that the program is not using a GOTO where there should be a GOSUB. The NEXT part of a FOR . . . NEXT loop could not find the matching FOR. Check for incorrect variables after the NEXT and for incorrectly nested loops.

BAD VALUE	The number used is incorrect. First check the command that the number is used in conjunction with. Be sure that you are not using a number too large or too small for the command. Check the sign of the number. Some commands will not accept negative numbers or a zero.
INCORRECT STATEMENT	There is a definite problem in the program line, and the computer cannot execute the program any further. Check the program line for missing commands, parentheses, variables, arithmetic signs, and line numbers. Reserved words cannot be used as variables.

These are some of the most common errors you can get from a BASIC program. There are other error messages related to the use of the disk drives, printers, and other accessories.

Chapter 6

Storing and Accessing the Program

Programs can be stored on cassettes or floppy disks. This chapter discusses only the commands used to store and load programs on the cassette recorder.

Your TI-99/4A can have two cassette recorders connected to it. The first, the number 1 cassette recorder, can be used to load a program into the computer or save a program that is in the computer onto the cassette. The second, the number 2 cassette recorder, can only be used to save programs.

OLD

To get a program from a cassette tape into the computer, insert the cassette in the recorder, make sure that it is properly positioned, type OLD CS1, and press the ENTER key. The computer gives you instructions on the screen. Follow these instructions carefully. First, you are instructed to rewind the tape that is in the first cassette recorder. If the program that you want to load is the first program on the tape, then rewind the cassette. If it is the second or third program on the tape, and you are sure that you have the tape correctly positioned, you can omit this step and press ENTER.

Now press the PLAY button on the cassette recorder and the ENTER key on the computer. The screen displays "READING." The computer listens to the tape and converts the tones that it hears into the program instructions. If the tape loads successfully, the cursor appears on the screen. Type RUN and press ENTER to begin the program.

If the computer cannot read the program correctly, an error message will appear on the screen and you are given the option of trying the reading procedure again or stopping. To read again, press the R key. Be sure that you enter it in uppercase. Use the shift key if necessary. If you do not want to try again, press the E and another error message is displayed on the screen. Sometimes an error message means that the tape has a defect in it. Other times you may have placed the wrong tape or blank tape in the recorder or not positioned it correctly. It may also mean that the heads on the tape recorder are dirty and should be cleaned.

SAVE

Once you have typed a program into the computer, you will want to store it before shutting off the machine. The SAVE command places the program in RAM onto a cassette. To save your program, type SAVE CS1 and press the ENTER key. The computer displays instructions on the screen to help you save your program correctly. The same instruction is displayed if you are using a second recorder and CS2 as the recorder number.

First you are instructed to rewind the cassette tape. Again, if this is the second or third program that you are saving on the tape, it is not necessary to rewind the tape. Press ENTER to continue. Press the RECORD and PLAY buttons on your recorder and press the ENTER key again. The screen displays RECORDING. The computer converts the program instructions into tones and sends these tones to the recorder. The tones represent the binary numbers that the computer converts back into instructions when the program is read back into the computer.

Once the program has been saved, the computer instructs you to press the STOP button on your recorder and press the ENTER key. You are then given the option of checking the tape for the program that you just saved. Press Y. You are instructed to rewind the tape, press the ENTER key, press the PLAY key on the recorder, and the ENTER key on the computer. If the program has been saved correctly, the screen displays DATA OK. This is a very good feature. If the program has not been saved correctly, or was not saved at all because both the PLAY and RECORD buttons were not pressed on the recorder, you have not lost your program. Your original program is still in the computer and you can save it again.

Once you have saved the program correctly, the cursor appears in the lower corner of the screen, and you can continue programming, run the program, or quit.

PROTECTED

In TI Extended BASIC there is a *protected* option that is available for use with programs that you do not want to change while they are being used. When you use the SAVE command, add ,PROTECTED to the command as shown here:

SAVE CS1,PROTECTED {press ENTER}

The computer saves the program to the cassette. When the program is loaded back into the computer, the user can RUN the program, but cannot edit, list, or resave it. A word of caution here, always save an unprotected copy of your final program just in case you will want to list or edit the program at a later date.

RUN

This command is used to begin a program. When the command is used with no numbers following it, the computer begins at the first line of the program. All the variables are set to zero, the strings are cleared so that they are empty, and if any area of memory was used for special graphics characters, that space is also cleared.

You can also start the program at any line number. All variables will be cleared. If you try to run a program at a line number that does not exist, the computer will give you an error message. If you try to RUN a program when none exists, the computer will tell you that it CAN'T DO THAT.

Chapter 7

Understanding the Screen

Your TI-99/4A can display letters, numbers, and characters anywhere on your screen. Your screen size or *resolution* is 32 characters across and 24 lines high. This resolution is referred to as a 32×24 screen. Each character on the screen is made up of dots called *pixels*. These pixels are turned on to form the letters or characters. The characters use 8 pixels across and 8 pixels down. The actual resolution of the screen is 256×192 because that's how many pixels there are. We will work with the pixels and high resolution graphics in a later chapter. In this chapter, we will work with the standard set of characters that are available with your TI-99/4A.

DISPLAYING THE PROGRAM

After you have loaded a program into memory, you may want to look at it to see which commands are used, or to change the line instructions. You can look at a program by typing LIST and pressing ENTER. The entire program will be printed on the screen. If the program is longer than 23 lines, the first lines will scroll off the tip of the screen. Unless you can speed read, the program will scroll by too fast. To pause the listing, hold down SHIFT and press the S key. The computer will stop the listing at the end of that line. To continue the listing, press any key.

You may also tell the computer to list only the lines that you would like to read. LIST 10-50 tells the computer to start with line 10 and list all the program statements up to and including line 50. If there are more lines than can fit on the screen, the first lines will scroll off. You can also use LIST -50 to list all the lines in the program from the first line, up to and including line 50. LIST 50- will list all the lines beginning with line 50 to the last program statement. LIST 50 would only list program statement 50.

To tell the computer to execute the program in its memory, type RUN and press ENTER. The computer will start with the first line of the program and complete the instructions in that line; then proceed to the next line and follow those instructions. Should you want to stop a program after one

section has run, you can press FCTN and the 4 key. The screen will display:

*BREAKPOINT IN {line number}

You may put in several breakpoints to help you debug your program and remove them later.

SIZE

Ever wonder how much memory a program uses, or how much room (free RAM) you have left? Type SIZE and press ENTER to find out how much memory (RAM) is left. Everything that is used in a program uses memory (RAM). This includes the screen, string and numeric variables, sprite and color tables, and of course, the program. If you have the Memory Expansion unit attached to your TI-99/4A, the computer stores the program, variable, and table information differently. The SIZE command tells you how much program memory is available as well as how much stack space is free. The program memory holds the program and the numeric variables. All other information is stored in the stack space.

Always run the program to get the true amount of free memory. Some programs set aside some of the memory for storage and this is not evident until you run the program.

PRINTING TO THE SCREEN

The video screen is the primary display for your program. Even though you can use a printer, voice synthesizer, or other accessories with your TI-99/4A, most programs are presented on the video screen. You should try to keep unrelated information off the screen when you are running your program. When you begin to write a program, the first few lines should be remark lines with information about the program. The first program line should clear the screen. The format for clearing the screen during a program is as follows:

```
50 CALL CLEAR
```

This command will remove any previous information from the screen. Directions can then be printed on the screen for the user to read while the computer is setting up the program.

Displaying words or characters to the screen is accomplished with the PRINT command.

```
60 PRINT "ANYTHING YOU WANT"
```

The computer places whatever is between the quotation marks on the screen. There must be quotation marks before and after the words or characters that you want printed. If there are several lines to be printed in a program, each new line will be displayed under the previous one.

There may be times when you will want several different items printed on the same line with or without spaces between them, for example, columns with headings above each column. Two characters, when placed at the end of a line, will hold the cursor in the same line—the comma and the semicolon.

The semicolon will not advance the cursor after the last character of a print statement has been printed. The first character of the next print statement will occupy the next position on the screen.

```
60 PRINT "HELLO";
70 PRINT "THERE"
```

If you run this two-line program, your screen should display:

```
HELLOTHERE
```

There is no space between the "O" in "hello" and the "T" in "there." The semicolon indicated no space, so the next word began in the next position on the same line.

There is one case where the semicolon works differently. If the next string, whether it is a set of characters between quotation marks or a string variable, is too long to fit on that same line, it will be printed on the next line. Try this:

```
60 PRINT "THIS IS THE FIRST SENTENCE";
70 PRINT "THIS IS THE SECOND
   SENTENCE!"
```

Run this two-line program. As you can see, the sentence in line 70 cannot possibly fit on the same line as the sentence in line 60, so the computer moves it to the next line on the screen.

A comma functions somewhat differently from the semicolon. It will place the next string in a particular column on the screen. Try these lines:

```
60  PRINT "DATE",
70  PRINT "PLACE",
80  PRINT "TIME"
```

Run these lines. The screen should display:

```
DATE        PLACE
TIME
```

There are two distinct columns on the screen. The comma indicates that the next line should print in the next available column. Since there is no comma after the word "TIME," the next print statement would place its information on the next line under the word "TIME."

Printing numbers is a little different. Numbers do not have to be enclosed in quotes. Negative numbers always display the minus sign. With positive numbers, the plus sign is understood. When these numbers are printed on the screen, the computer leaves a space before a positive number, and a space after all numbers whether they are negative or positive, so that numbers with a semicolon between them do not look like one long number when printed on the screen. Try this example:

```
 60  PRINT 1;
 70  PRINT -1;
 80  PRINT -2;
 90  PRINT 2;
100  PRINT 3;"!"
```

When this program is run, the screen will look like:

```
 1 -1 -2  2  3 !
```

There is one space between the left edge of the screen and the first "1." This is the understood plus sign. There is one space after the first "1." This is the trailing space for the number. There is one space between the "-1" and the "-2." After the "-2," there are two spaces. One is the trailing space, the other is the understood positive sign for the "2." There is one space between the "3" and the exclamation mark (!).

There are times when neither the comma or the semicolon will place the information in the correct position on the line. Any horizontal location can be addressed by using the TAB command with the PRINT command, as shown below.

```
PRINT TAB(20);"HELLO"
```

This program line will begin to print the word "hello" in the twentieth space or column of that line. On the other hand,

```
PRINT "HELLO";TAB(20);"THERE"
```

will print the word "hello" at the beginning of the line, then print the word "there" at the twentieth column. When you need more than two columns on the screen, use the TAB command.

DISPLAY

Used by itself, the DISPLAY command is similar to the PRINT command. The information following it is printed on the screen. Used with the word AT, you can print information anywhere on the screen. There are several options that you can choose from when you use the DISPLAY AT command. Following the AT, specify the number of the row and column that your information should be printed at.

```
70  DISPLAY AT(4,8):"4TH ROW - 8TH
    COLUMN"
80  DISPLAY AT(10,15):"10TH ROW - 15TH
    COLUMN"
```

Before you run these two lines, be sure that all other program lines have been deleted. After you run these lines, you should see both lines on your

screen, each in its correct position, as indicated by the two numbers in the parentheses. The first number indicates how many rows or lines down from the top of the screen this information will be printed on. The second number is the column, or how far in from the left side of the screen the information will be. (There are 24 rows and 28 columns that can be accessed with the DISPLAY AT command.) If you tell the computer to print something outside of that range, for example: DISPLAY AT(30,2):"HI", a #79 - BAD VALUE error message will result.

The DISPLAY AT(r,c) BEEP command will make a sound before the message is printed on the screen.

```
DISPLAY AT(2,3)BEEP:"SOUND"
```

The SIZE option used with the DISPLAY AT command places spaces (or erases) a certain number of positions on the screen. The number of spaces to be printed on the screen is placed within the parentheses after the SIZE command.

```
DISPLAY AT(2,3)SIZE(2)
```

Starting at location 2,3 two blank spaces across will be displayed.

To clear the entire screen, you can use the ERASE ALL option. Used with the DISPLAY AT command, you can have a message printed on the screen immediately after the screen is cleared.

```
DISPLAY AT(3,4)ERASE ALL:"HELLO AGAIN"
```

If the message to be printed on the screen will NOT fit on the line that you want it to, the computer will move it down one line and ignore the column number. If your message exceeds one line length, part of the message will be on the next line. Try this:

```
DISPLAY AT(7,4):"THIS MESSAGE IS
OBVIOUSLY TOO LONG TO FIT ON ONE LINE"
```

The message does not begin at the fourth column of the seventh row, but at the first column of the eighth row and continues on the ninth row.

The following programs will give you some ideas on how to use the PRINT and DISPLAY AT commands. The commands not yet introduced will be covered later in this book. Listing 7-1 is flowcharted in Fig. 7-1.

Listing 7-1

Line 130 clears the screen with the CALL CLEAR subroutine.

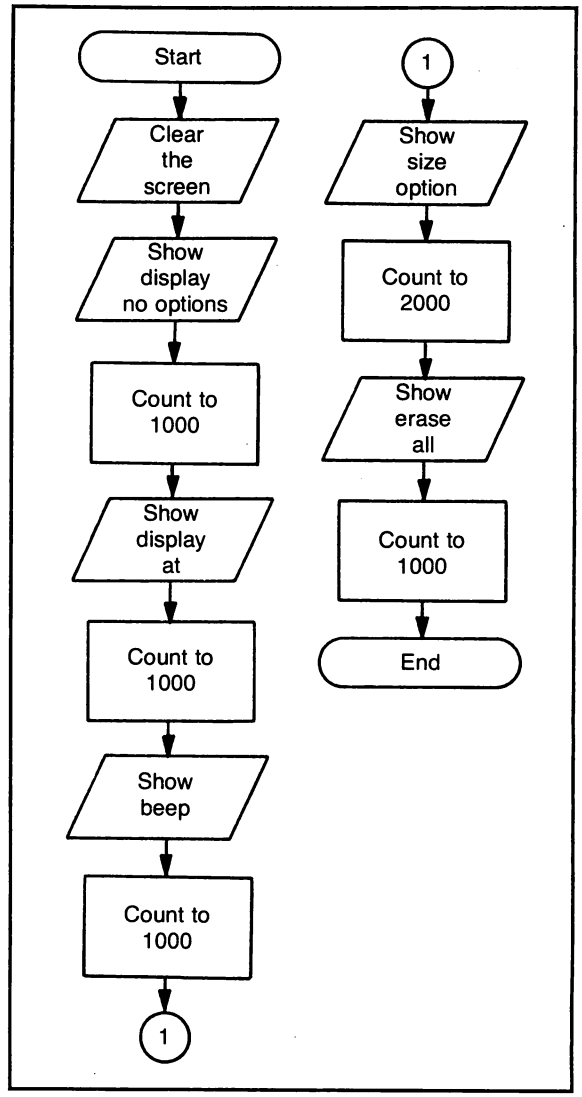


Fig. 7-1. Flowchart for Listing 7-1 Display Options.

Listing 7-1

```
100 REM LISTING 7-1
110 REM DISPLAY EXAMPLES
120 REM A.R.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR ! CLEAR SCREEN
140 REM USE DISPLAY AS PRINT
150 DISPLAY :"'DISPLAY' WITHOUT OPTIONS
WORKS THE SAME AS 'PRINT'"
160 FOR DELAY=1 TO 1000 :: NEXT DELAY
170 REM DISPLAY WITH "AT"       OPTION
180 DISPLAY AT(12,2):"WITH AT OPTION ROW
 12,COL 2"
190 FOR DELAY=1 TO 1000 :: NEXT DELAY
200 REM WITH BEEP OPTION
210 DISPLAY AT(14,5)BEEP:"WITH BEEP !!"
220 FOR DELAY=1 TO 1000 :: NEXT DELAY
230 REM WITH SIZE OPTION
240 DISPLAY AT(12,7)SIZE(18)BEEP:"SIZE O
PTION AT 18"
250 FOR DELAY=1 TO 2000 :: NEXT DELAY
260 REM WITH ERASE ALL OPTION
270 DISPLAY AT(12,5)ERASE ALL BEEP:"WITH
 ERASE ALL OPTION"
280 FOR DELAY=1 TO 1000 :: NEXT DELAY
```

Line 150 demonstrates using the DISPLAY command like a PRINT command.

Line 160 is a delay loop. This slows down the program so that you can watch the messages appear on the screen.

Line 180 uses the DISPLAY AT command. This message begins at the second column of the twelfth row on the screen.

Line 210 uses the BEEP option. The computer sounds a short beep. Then the words "with beep" are printed at the fifth column of the fourteenth row.

Line 240 uses the SIZE option to erase 18 letters from the twelfth row beginning with the seventh column. The computer beeps; then the words "size option at 18" replace the letters that were erased from that line.

Line 270 erases the entire screen with the ERASE ALL option. The computer beeps and the message is printed on the screen.

Listing 7-2 displays a LOVE graphic. This is created by using the SIZE command with the DISPLAY AT command to erase characters from the screen. After the remarks in lines 100 to 120 the computer follows the steps flowcharted in Fig. 7-2.

Listing 7-2

Line 130 clears the screen.

Lines 150-170 use a new command. The RPT$ command tells the computer to use the string "LOVE" 63 times; then 63 times; then 42 times. Since the RPT$ command can only print 255 characters at one time, it is necessary to divide the pattern into three program lines.

Lines 190-300 use the DISPLAY AT with the SIZE command to form the letter "L." Each line specifies a new row and column that will have letters erased. The number after SIZE is the number of characters that will be erased.

Lines 320-470 erase letters to form the letter "O."

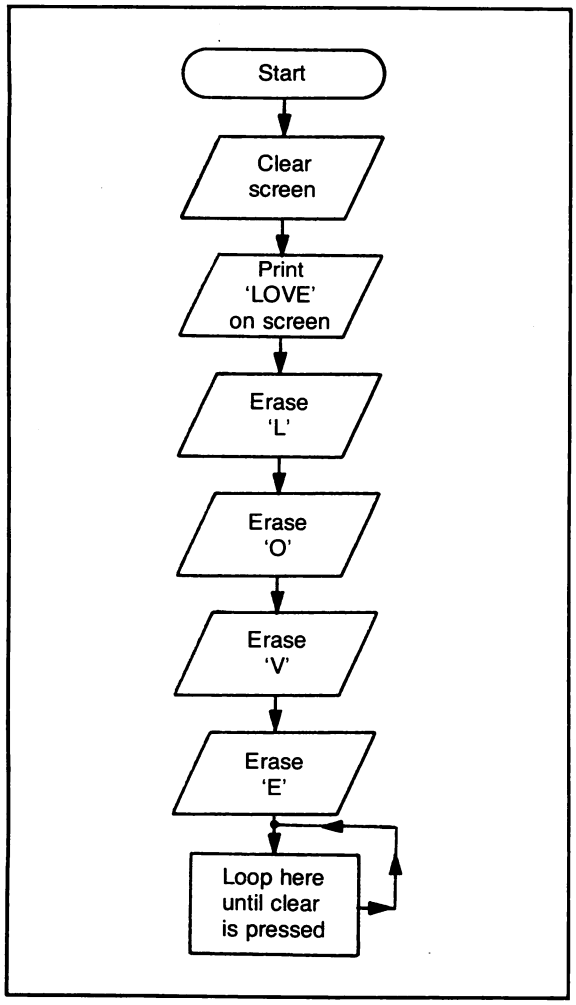


Fig. 7-2. Flowchart for Listing 7-2 LOVE pattern.

Lines 490-680 erases letters to form the "V."
Lines 700-840 form the letter "E."
Line 860 does not end the program with an END statement that would tell the computer to display the cursor at the bottom of the screen and part of the display would be lost. Instead, we use the GOTO command. This returns the computer to the beginning of the line. This is the only command on this line, so the computer continues to go to line 860 until you press FCTN and 4 to stop the program.

Listing 7-3 demonstrates simple animation. The letters are printed in a specific location, erased, and reprinted in a new position. (See flowchart in Fig. 7-3.) Entire words can be moved across the screen this way.

Listing 7-3

Line 130 clears the screen.
Line 140 is the beginning of a FOR . . . NEXT loop. This command saves memory and typing. The X variable refers to the column the computer will be printing the information in.
Line 150 positions the cursor at the point on the screen at which we want a letter printed. X indicates the column that the "M" will be printed in. There is a space before the letter "M." Before the computer prints the "M," it will print the space. This erases the previous "M" that was printed on the screen and gives the illusion of an "M" moving across the screen.

Listing 7-2

```
100 REM LISTING 7-2
110 REM LOVE
120 REM L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR ! CLEAR SCREEN
140 REM FILL SCREEN WITH LOVE
150 PRINT RPT$("LOVE",63);
160 PRINT RPT$("LOVE",63);
170 PRINT RPT$("LOVE",42);
180 REM MAKE AN 'L'
190 DISPLAY AT(2,2)SIZE(7)
```

```
200 DISPLAY AT(3,2)SIZE(7)
210 DISPLAY AT(4,4)SIZE(3)
220 DISPLAY AT(5,4)SIZE(3)
230 DISPLAY AT(6,4)SIZE(3)
240 DISPLAY AT(7,4)SIZE(3)
250 DISPLAY AT(8,4)SIZE(3)
260 DISPLAY AT(9,4)SIZE(3)
270 DISPLAY AT(9,12)SIZE(2)
280 DISPLAY AT(10,3)SIZE(11)
290 DISPLAY AT(11,3)SIZE(11)
300 DISPLAY AT(12,3)SIZE(11)
310 REM MAKE AN 'O'
320 DISPLAY AT(2,19)SIZE(7)
330 DISPLAY AT(3,19)SIZE(7)
340 DISPLAY AT(4,17)SIZE(11)
350 DISPLAY AT(5,17)SIZE(3)
360 DISPLAY AT(5,25)SIZE(3)
370 DISPLAY AT(6,17)SIZE(3)
380 DISPLAY AT(6,25)SIZE(3)
390 DISPLAY AT(7,17)SIZE(3)
400 DISPLAY AT(7,25)SIZE(3)
410 DISPLAY AT(8,17)SIZE(3)
420 DISPLAY AT(8,25)SIZE(3)
430 DISPLAY AT(9,17)SIZE(3)
440 DISPLAY AT(9,25)SIZE(3)
450 DISPLAY AT(10,17)SIZE(11)
460 DISPLAY AT(11,19)SIZE(7)
470 DISPLAY AT(12,19)SIZE(7)
480 REM MAKE A 'V'
490 DISPLAY AT(14,2)SIZE(3)
500 DISPLAY AT(14,11)SIZE(3)
510 DISPLAY AT(15,2)SIZE(3)
520 DISPLAY AT(15,11)SIZE(3)
530 DISPLAY AT(16,3)SIZE(3)
540 DISPLAY AT(16,10)SIZE(3)
550 DISPLAY AT(17,3)SIZE(3)
560 DISPLAY AT(17,10)SIZE(3)
570 DISPLAY AT(18,4)SIZE(3)
580 DISPLAY AT(18,9)SIZE(3)
590 DISPLAY AT(19,4)SIZE(3)
600 DISPLAY AT(19,9)SIZE(3)
610 DISPLAY AT(20,5)SIZE(3)
620 DISPLAY AT(20,8)SIZE(3)
630 DISPLAY AT(21,5)SIZE(3)
```

```
640 DISPLAY AT(21,8)SIZE(3)
650 DISPLAY AT(22,6)SIZE(3)
660 DISPLAY AT(22,7)SIZE(3)
670 DISPLAY AT(23,7)SIZE(2)
680 DISPLAY AT(24,7)SIZE(2)
690 REM MAKE AN 'E'
700 DISPLAY AT(14,17)SIZE(11)
710 DISPLAY AT(15,17)SIZE(11)
720 DISPLAY AT(16,18)SIZE(3)
730 DISPLAY AT(16,26)SIZE(2)
740 DISPLAY AT(17,18)SIZE(3)
750 DISPLAY AT(18,18)SIZE(3)
760 DISPLAY AT(18,24)SIZE(1)
770 DISPLAY AT(19,18)SIZE(7)
780 DISPLAY AT(20,18)SIZE(7)
790 DISPLAY AT(21,18)SIZE(3)
800 DISPLAY AT(21,24)SIZE(1)
810 DISPLAY AT(22,18)SIZE(3)
820 DISPLAY AT(22,26)SIZE(2)
830 DISPLAY AT(23,17)SIZE(11)
840 DISPLAY AT(24,17)SIZE(11)
850 REM LOOP HERE SO PROGRAM WON'T END U
NTIL (CLEAR) IS PRESSED
860 GOTO 860
```

Line 160 finishes the loop. The computer completes lines 140 to 160 six times before it goes on the next line.

Lines 170-190 move the "O" across the screen in a similar manner.

Line 200 erases the "O" from the line that it was moving on.

Line 210 moves the "O" up one line, placing it immediately after the "M."

Lines 220-240 move the "V" across the screen above the letters "MO."

Line 250 erases the "V" from the line above the "MO."

Line 260 prints the "V" after the "MO."

Lines 270-410 follow the same pattern of printing a letter on the left side of the screen, erasing it, and printing it one column over.

Lines 420-440 move an entire word across the screen. Again, there is a space before the word WORDS. This space erases the letter "W" in the word previously printed. If there were no space, a line of "Ws" would be printed across the screen.

Lines 450-500 use the SIZE command to erase the entire word and print it one line lower.

Line 510 ends the program. The message "MOVING WORDS" should be on the screen, along with the prompt.

Although this program moved only letters across the screen, using this technique of printing and erasing you can use graphics created with new characters and move them anywhere on the screen.

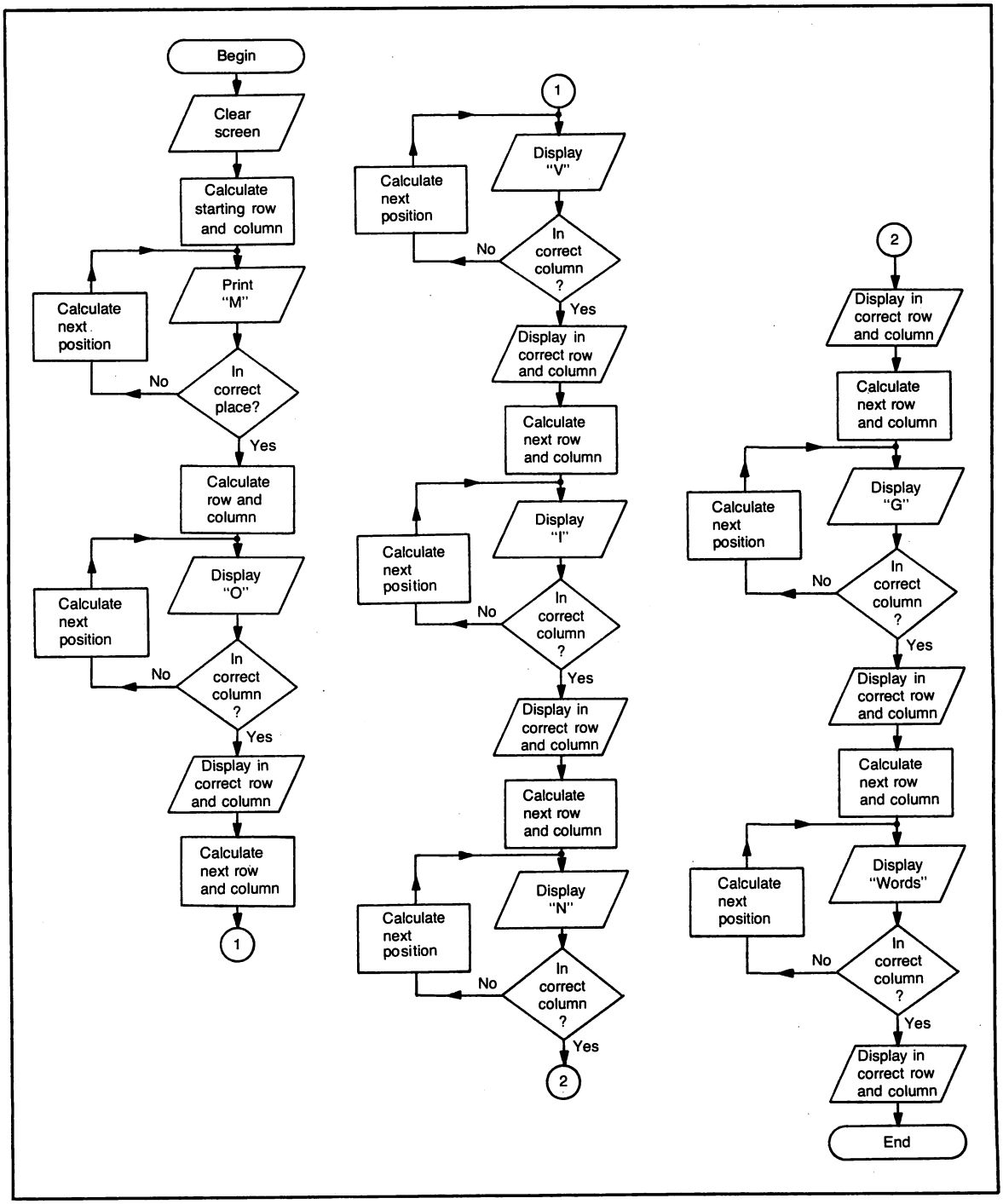


Fig. 7-3. Flowchart for Listing 7-3 Simple Animation.

Listing 7-3

```
100 REM LISTING 7-3
110 REM SIMPLE ANIMATION
120 REM L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR ! CLEAR SCREEN
140 FOR X=1 TO 6 ! THIS COMMAND SAVES ME
MORY & TYPING
150 DISPLAY AT(5,X):" M" ! CHANGE THE CO
LUMN BUT NOT THE ROW
160 NEXT X ! DO IT 6 TIMES
170 FOR X=1 TO 7
180 DISPLAY AT(6,X):" O" ! NOW DO IT ONE
 ROW LOWER
190 NEXT X
200 DISPLAY AT(6,7):" " ! ERASE IT
210 DISPLAY AT(5,8):"O" ! MOVE IT UP
220 FOR X=1 TO 8
230 DISPLAY AT(4,X):" V"
240 NEXT X
250 DISPLAY AT(4,8):" "
260 DISPLAY AT(5,9):"V"
270 FOR X=1 TO 9
280 DISPLAY AT(6,X):" I"
290 NEXT X
300 DISPLAY AT(6,9):" "
310 DISPLAY AT(5,10):"I"
320 FOR X=1 TO 10
330 DISPLAY AT(4,X):" N"
340 NEXT X
350 DISPLAY AT(4,10):" "
360 DISPLAY AT(5,11):"N"
370 FOR X=1 TO 11
380 DISPLAY AT(6,X):" G"
390 NEXT X
400 DISPLAY AT(6,11):" "
410 DISPLAY AT(5,12):"G"
420 FOR X=1 TO 16 ! MOVE IT ACROSS
430 DISPLAY AT(2,X):" WORDS"
440 NEXT X
450 DISPLAY AT(2,17)SIZE(5)! ERASE IT
460 DISPLAY AT(3,17):"WORDS"
470 DISPLAY AT(3,17)SIZE(5)
480 DISPLAY AT(4,17):"WORDS"
490 DISPLAY AT(4,17)SIZE(5)
500 DISPLAY AT(5,17):"WORDS"
510 END
```

Chapter 8
Getting the Answers

Sometimes a value that the program needs will change every time the program is run. Sometimes a new value will be entered by the computer user, other times the programmer wants the computer to use a value different than the one previously calculated. The computer needs to be able to keep track of the value by storing it in a place in memory so it can *recall* the value when it needs to. To accomplish this, the program stores the value as a *variable*.

ASSIGNING VALUES

LET

A variable is a letter, group of letters, or word that represents a value. If you were to enter

```
20  LET A=10
```

the computer would substitute the value 10 each time it encountered the A variable in the program. If the program contained

```
30  LET B=A+5
```

it would add 10 to 5. The B variable would become 15. The LET command is used to set a variable to a value. The command does not have to be used. It is an understood command, so you could enter

```
30  B=A+5
```

If you have a program in your computer, type NEW and then type in Listing 8-1. (See flowchart in Fig. 8-1.)

Listing 8-1

Line 140 sets the A variable equal to 10.
Line 150 sets the B variable equal to 15.
Line 160 sets the C variable equal to 20.
Line 170 prints all three variables on the same line.

As you can see, when the computer is told to print a variable, it prints the value that it stored in that variable. Your screen should display:

```
10  15  20
```

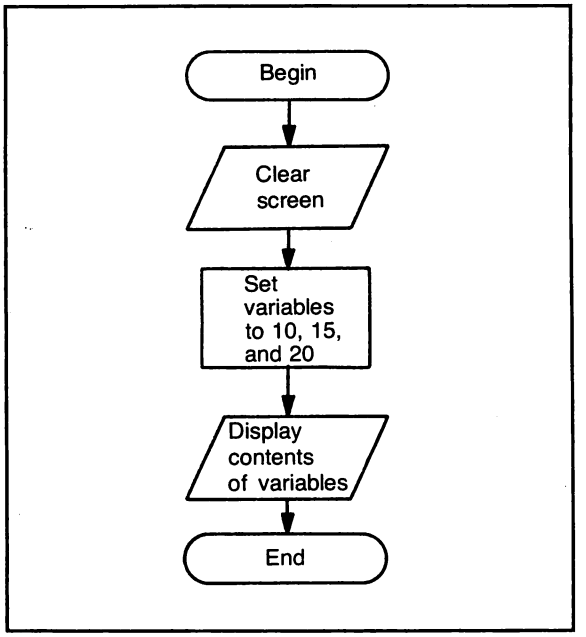


Fig. 8-1. Flowchart for Listing 8-1 Assign Variable Values.

The value of a variable can be changed and reused throughout the program. A variable can also be used instead of a number for an arithmetic operation. Type NEW and enter the program shown in Listing 8-2. (Also see flowchart in Fig. 8-2.)

Listing 8-2

Line 140 assigns the LENGTH variable a value of 30 and the WIDTH variable a value of 7. Names can be used as variables.

Line 150 prints a message on the screen. Since the value of the variable will be printed on the same line, a semicolon is placed after the quotation mark and before the variable. There is no need to place a space between the last letter of the last word and the quotation mark since there will be a leading and trailing space when the number is printed. A semicolon is placed after the LENGTH variable so that the next word of the message will be on the same line.

Line 160 performs the calculation that determines the perimeter of the room. The answer is stored in the PERIMETER variable.

Line 170 contains two print statements. The first PRINT command is *not* followed by a message; it brings the cursor down one line so that the message contained after the second PRINT command will be one row below the last message. A print statement by itself skips a line on the screen. This message will be printed on the next line of the screen.

Line 180 changes the values of the LENGTH and WIDTH variables. Whenever possible, the same variables should be reused in a program. This saves memory and is efficient. Each time a variable is assigned a new value, the computer forgets the old value.

Line 190 skips a line on the screen, then prints another message on the screen. The colons between parts of the message tell the computer to use a new line for that portion of the message.

Line 200 calculates the area of the room and stores

Listing 8-1

```
100 REM LISTING 8-1
110 REM ASSIGN VARIABLES VALUES
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR ! CLEAR SCREEN
140 A=10 ! THE VARIABLE 'A' WILL BE 10
150 B=15 ! THE VARIABLE 'B' WILL BE 15
160 C=20 ! THE VARIABLE 'C' WILL BE 20
170 PRINT A;B;C ! THE SEMI-COLONS WILL K
EEP THE VALUES ON THE SAME LINE
180 END
```

Listing 8-2

```
100 REM LISTING 8-2
110 REM CHANGE VARIABLES VALUES
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR ! CLEAR SCREEN
140 LENGTH=30 :: WIDTH=7 ! NAMES CAN BE
VARIABLES
150 PRINT "I will now compute the":"peri
meter of a room that hasa lensth of";LEN
GTH;"and a width":"of";WIDTH
160 PERIMETER=2*WIDTH+2*LENGTH ! FORMULA
 FOR PERIMETER
170 PRINT :: PRINT "THE PERIMETER IS";PE
RIMETER
180 LENGTH=27 :: WIDTH=14 ! CHANGE THE V
ALUES OF THE VARIABLES
190 PRINT :: PRINT "Now I will calculate
the":"area of a room whose width":"is";
WIDTH;"and whose lensth is";LENGTH
200 AREA=WIDTH*LENGTH ! FORMULA FOR AREA
210 PRINT :: PRINT "THE AREA IS";AREA
220 END
```

the answer in the AREA variable.

Line 210 skips a line on the screen, then prints the message containing the area of the room.

It is important to note that there *must* be a semicolon or comma between the message and the variable in the print statements. Without semicolons, BASIC will not accept the line. To print the value of the variable on the next screen line, use a colon instead of a semicolon.

MAX/MIN

The MAX command compares two numbers or variables to see which one is larger. There are many different uses for this command when you are writing a program that needs to know which variable is the largest.

The MIN command is just the opposite. It determines which variable or number is smaller. The formats for these commands are:

```
70  X=MAX(B,C)    or
70  X=MAX(10,20)  or
70  X=MAX(B,10)

80  X=MIN(B,C)    or
80  X=MIN(10,20)  or
80  X=MIN(B,10)
```

The program in Listing 8-3 (flowcharted in Fig. 8-3) is a routine from the end of a game where the program checks the scores of both players to see if the previous high score has been beaten. It shows how the MAX command can be used.

Listing 8-3

Line 130 clears the screen, then prints the message HI SCORE in the fourth row beginning with the column 14.

Line 140 sets the HISCR variable to 758 and the CRNTSCR variable or 902. These variables could be any score, but have been set here for this example.

Line 150 prints the value of HISCR immediately

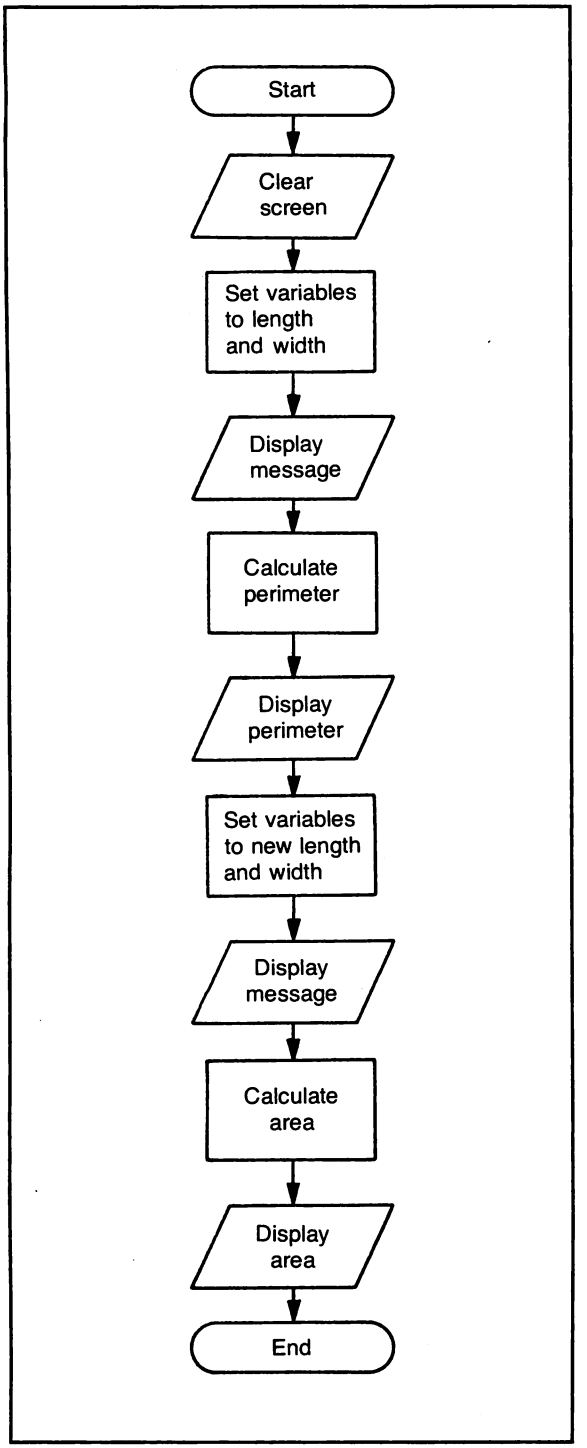


Fig. 8-2. Flowchart for Listing 8-2 Change Variable Values.

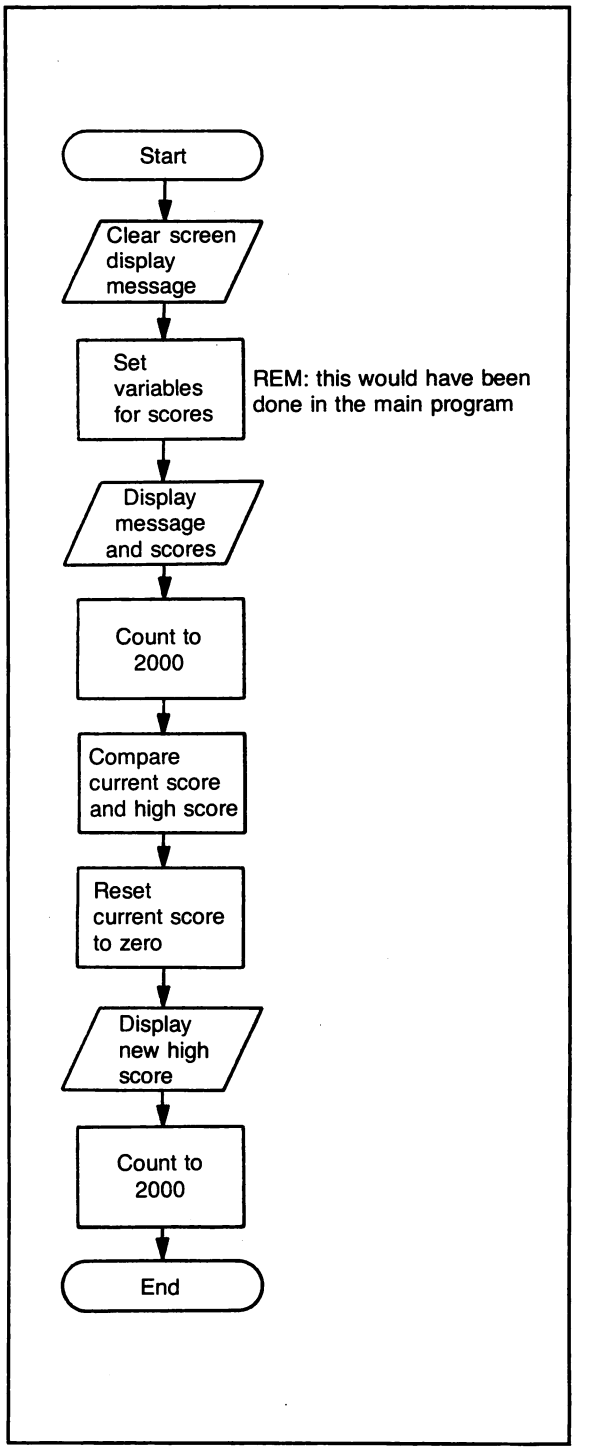


Fig. 8-3. Flowchart for Listing 8-3 MAX Example.

Listing 8-3

```
100 REM LISTING 8-3
110 REM MAX EXAMPLE
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: DISPLAY AT(4,14):"HI S
CORE :"
140 HISCR=758 :: CRNTSCR=902
150 DISPLAY AT(4,24):HISCR :: DISPLAY AT
(14,9):"GAME OVER" :: DISPLAY AT(24,17):
"SCORE ";CRNTSCR
160 FOR DELAY=1 TO 2000 :: NEXT DELAY
170 HISCR=MAX(HISCR,CRNTSCR):: CRNTSCR=0
180 DISPLAY AT(4,24)BEEP:HISCR :: DISPLA
Y AT(14,9)BEEP:" NEW GAME" :: DISPLAY AT
(24,23)BEEP:CRNTSCR
190 FOR DELAY=1 TO 2000 :: NEXT DELAY
```

after the words HI SCORE on the screen. This is the current high score for this game. At the fourteenth row, GAME OVER is displayed. The score for this game is shown in row 24. The computer will beep before the current score is printed.

Line 160 is a timing loop. It waits for a few seconds so you can read the screen.

Line 170 compares the current score to the high score. The larger value will be stored in the HISCR variable. The variable for the current score will be reset to zero for the next game.

Line 180 changes the score. Now the new high score is on the screen. The message at row 14 is changed to NEW GAME and the current score of zero is displayed after the word SCORE.

Line 190 is another delay loop to give you a chance to read the screen. The program ends after this line.

USING STRING VARIABLES

Numeric variables store numbers, but *string variables* can store a "string" of numbers, letters, or characters. However, string variables cannot be used in arithmetic functions even though they may store numbers. Numbers that you will not be adding, such as dates or ID numbers are best housed in strings.

Like numeric variables, string variable names can be letters, groups of letters, or words. By placing a "$" at the end of a variable name, you are telling the computer that this is a string variable. Just like numeric variables, the contents of string variables can be printed on the screen. (See the flowchart in Fig. 8-4.)

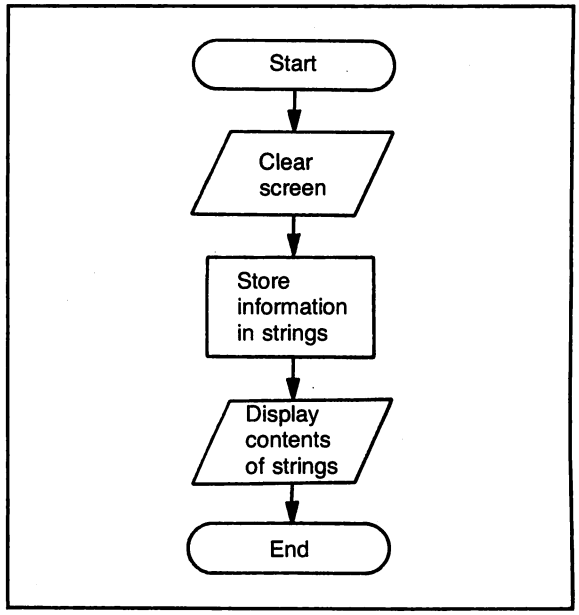


Fig. 8-4. Flowchart for Listing 8-4 String Variable Value.

Listing 8-4

```
100 REM LISTING 8-4
110 REM STRING VARIABLES VALUES
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR ! CLEAR SCREEN
140 NAME$="J.Q.PUBLIC" ! STORE NAME IN S
TRING
150 ADDRESS$="123 MAIN STREET" ! STORE A
DDRESS
160 CITY$="NEWTON" ! THIS IS CITY
170 STATE$="MI" ! USE THE TWO LETTER ABB
REVIATION
180 ZIP$="43201" ! ZIP CODE
190 DISPLAY AT(4,1):"THE ADDRESS INFORMA
TION IS:" ! SHOW WHAT IS STORED IN THE S
TRINGS
200 DISPLAY AT(6,7):NAME$
210 DISPLAY AT(8,7):ADDRESS$
220 DISPLAY AT(10,7):CITY$
230 DISPLAY AT(10,17):STATE$
240 DISPLAY AT(12,17):ZIP$
250 END
```

Listing 8-4

Lines 140-180 store information in each string. The letters, numbers, or characters that are placed in each string variable must be enclosed by quotation marks.

Lines 190-240 print the information in each string on the screen. DISPLAY AT places the information of each string at a particular place on the screen.

Like numeric variables, string variables hold their contents until they are changed by the program.

LEN

There are times when you need to know the length of a string. For example, you may want the title of your new program to be centered on the screen with the instructions printed under it. Your program would look like Listing 8-5 (flowchart in Fig. 8-5).

Listing 8-5

Line 140 places the title of the program in the string variable TITLE.

Line 150 uses the LEN command to find the length of the string. It places this information in variable L.

Line 160 divides the length of the title in half. We know that 28 letters can be placed on one row of the screen. If we subtract half of the length of the title from the center point of the screen (14), we will know where we should start printing the title.

Line 170 places the cursor at fourth row and in the P column. The title will be printed in the center of the screen.

Lines 180-190 begin the instructions for this program.

Line 200 is a loop that makes the computer wait until the FCTN and 4 keys are pressed.

If a program repeats itself or reuses some of

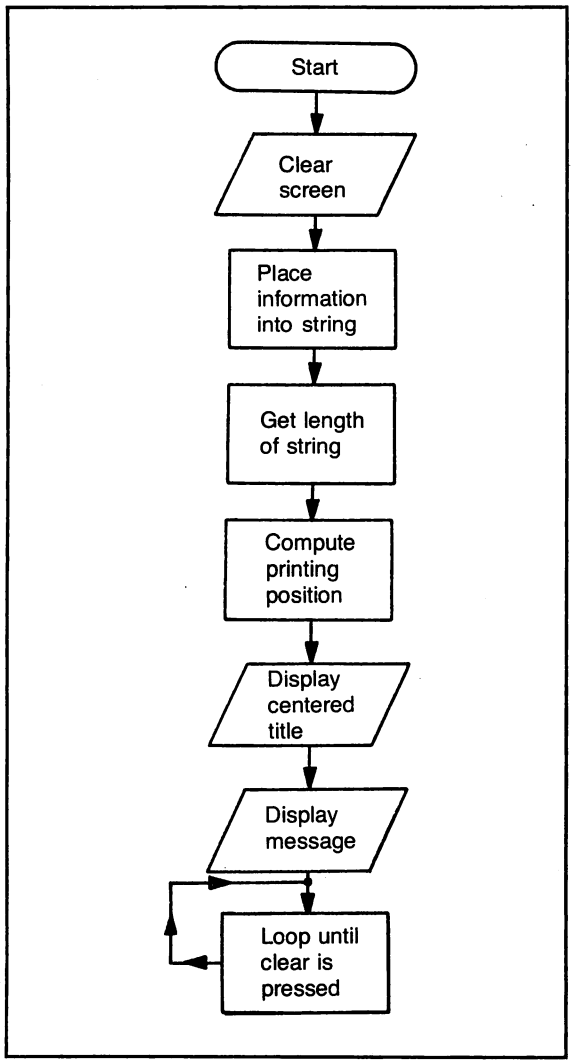


Fig. 8-5. Flowchart for Listing 8-5 Finding the Middle.

the variables, you can clear the previous information from a string by setting the string equal to " " (two quotation marks with nothing between them). This sets the string length to zero and erases all previous information. A string with no information in it is called a *null* string.

INPUT

So far, our variables and string variables have been assigned their value within the program. However, you don't always know the values ahead of time. The INPUT command allows you to enter a number, letter, or characters from the screen while the program is in use. If numbers are to be entered, then a numeric variable can be used for the entry. If letters or characters are entered, a string variable is used. In addition, your TI-99/4A can be set up so that only certain answers can be accepted. This is very useful when there can only be two answers to a question, and you don't want the computer to accept anything else. A True or False test is one example. The INPUT command can be used alone or with a message as a PRINT command.

```
20 INPUT B
```

If you ran this line it would display a question mark on the screen. The value entered would have to be a number and would be stored in the variable "B." One way to indicate what type of answer you want is to precede the variable with a question called a *prompt*.

```
20 INPUT"What is your name?":NAME$
```

Listing 8-5

```
100 REM LISTING 8-5
110 REM FINDING THE MIDDLE
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR ! CLEAR SCREEN
140 TITLE$="MORE SPACE WARS" ! GIVE THE
PROGRAM A TITLE
150 L=LEN(TITLE$)! FIND OUT HOW LONG THE
 STRING IS
160 P=14-L/2 ! GET HALF THE LENGTH AND S
```

```
UBTRACT IT FROM THE MIDDLE OF THE SCREEN
170 DISPLAY AT(4,P):TITLE$ ! 'P' IS THE
STARTING POSITION OF THE STRING
180 DISPLAY AT(6,1):"      This same requ
ires soodhand/eye coordination.   Youare
 the commander of a spaceship.   It  is
 beins  drawntoward"
190 DISPLAY AT(10,8):"another planet."
200 GOTO 200 ! STAY HERE UNTIL (CLEAR) I
S PRESSED
```

If you ran this line, the question would be printed on the screen. Your name would be stored in the string variable NAME$.

ACCEPT/VALIDATE/SIZE

An input can be accepted at a particular location on the screen. ACCEPT with the AT option is followed by the row and column number E.G. ACCEPT AT(22,28). The BEEP option can be used to get the user's attention.

The VALIDATE option is used after the ACCEPT command to specify which letters or numbers will be accepted for an entry. You can specify UALPHA for any uppercase letter, or DIGIT for any single-digit (0-9). NUMERIC allows the digits (0-9) plus the period, positive (+) and negative (−) signs, and the letter "E" for scientific notation. In addition, specific characters or letters can be enclosed in quotation marks, and those characters would be the only characters that the computer would accept. Any of these options can be used together. Here is an example of the use of some of these commands.

```
20 ACCEPT AT (2,2) VALIDATE (DIGIT,"Q")
```

The computer would accept a single-digit number or the letter "Q."

To accept more than one digit or letter, the SIZE option is used to specify the number that will be accepted. This option also clears that number of spaces at the ACCEPT AT position.

```
20 ACCEPT AT(2,2) VALIDATE(UALPHA)
   SIZE(5):ANSWER$
```

This program line clears five spaces beginning with row 2, column 2. The answer entered cannot be longer than five letters. This entry will be stored in ANSWER$.

If a negative number is used with the SIZE option, no spaces will be cleared at that location. The value of that number will determine how many characters will be accepted.

```
20 ACCEPT AT(2,2) VALIDATE(UALPHA)
   SIZE(−5):ANSWER$
```

This line is similar to the last line except no spaces will be cleared at location 2,2. The entry will still be limited to five characters.

The next program (Listing 8-6 and flowchart in Fig. 8-6) will compute the amount of sales tax to be added to an item and give the total price. The program demonstrates the commands discussed in this chapter.

Listing 8-6

Line 130 clears the screen.

Line 140 uses the INPUT command. The words in the quotation marks are printed on the screen as if the PRINT command were used. The program then waits until you enter a number. The program does not want the tax entered as a decimal or with the percent sign. If the tax rate in your state is

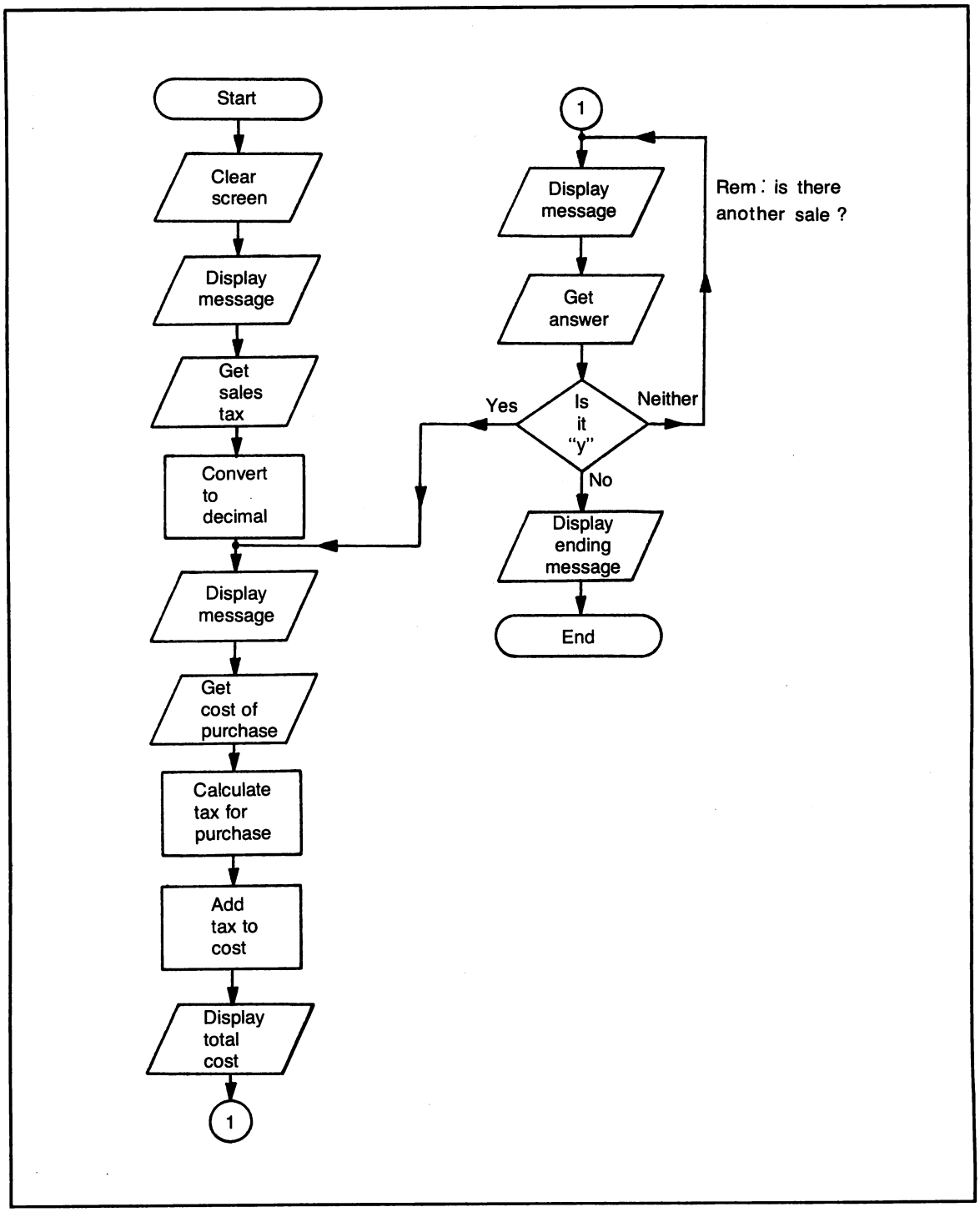


Fig. 8-6. Flowchart for Listing 8-6 Sales Tax.

Listing 8-6

```
100 REM LISTING 8-6
110 REM SALES TAX
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR
140 INPUT "ENTER YOUR STATE SALES TAX  (
NUMBERS ONLY, NO LEADING   DECIMALS OR P
ERCENT SIGN)    ?":TAX
150 TAX=TAX/100 ! CHANGE NUMBER ENTERED
TO A DECIMAL
160 PRINT :: INPUT "WHAT IS THE COST OF
THE     ITEM(S) ? ":COST
170 STAX=INT((COST*TAX+.005)*100)/100 !
ROUND TO NEAREST CENT AFTER MULTIPLYING
COST BY TAX
180 PRICE=COST+STAX ! TOTAL PRICE IS THE
 COST PLUS THE TAX
190 PRINT :: PRINT "THE COST OF THE ITEM
" :: PRINT "INCLUDING SALES TAX" :: PRIN
T "IS $";PRICE
200 PRINT : : : :: DISPLAY AT(22,2):"DO
YOU HAVE ANOTHER SALE?"
210 ACCEPT AT(22,28)BEEP VALIDATE("YN")S
IZE(1):ANSWER$
220 REM TEST FOR NO ANSWER
230 IF ANSWER$="" THEN 210
240 IF ANSWER$="Y" THEN 160
250 PRINT :: PRINT "HAVE A NICE DAY."
260 END
SOME ALTERNATE LINES FOR 200-210:
200 PRINT : : : :: DISPLAY AT(22,2):"DO
YOU HAVE ANOTHER SALE? Y"
210 ACCEPT AT(22,28)BEEP VALIDATE("YN")S
IZE(-1):ANSWER$
```

4%, then type the number "4" and press ENTER. The TAX variable will now be equal to the number that you entered.

Line 150 changes the number that you entered to a decimal by dividing the amount stored in TAX by 100.

Line 160 uses the PRINT command to place a space between the last message that was entered on the screen and the new one. The INPUT command is used again. The message between the quotation marks will be printed on the screen. The computer will wait until an amount has been typed, and ENTER has been pressed before going on to the next program line. Enter the cost of the item,

but do not enter a dollar sign. A period can be used as a decimal point.

Line 170 calculates the tax on the item. The state tax is the cost of the item times the tax. Since we are dealing with money, we want the tax to be rounded to the nearest penny. To do this, we add .005 to the amount arrived at *after* the cost of the item is multiplied by the tax rate. We then multiply the entire amount by 100. This shifts the decimal point two places to the right. If the tax came out to .473, adding .005 would change it to .478. Multiplying it by 100 would move the decimal to the right. The number would be 47.8. Now we take the integer (INT) of this number—that is, take only the whole number and ignore the decimal—and we have 47. Divide this by 100 and we have the tax of .47.

Line 180 adds the state tax to the cost of the item and stores it in the PRICE variable.

Line 190 uses a PRINT command to skip a line. The next PRINT command prints the total cost of the item including the sales tax.

Line 200 uses the PRINT command and three colons to place spaces on the screen. Be sure that this line is entered exact. There must be a space after the PRINT command, and after each of the three colons. The next two colons have no space between them. The DISPLAY AT command places the question at row 22, column 2.

Line 210 uses the ACCEPT AT command instead of the INPUT command. At the 22nd row and 28th column, the computer will beep. The VALIDATE option will only accept a "Y" or an "N" as an input. The SIZE(1) option erases one character at row 22, column 28. The entry will be stored in the string variable ANSWER$.

Lines 230-240 test the contents of ANSWER$. Since it is possible to press the RETURN key without pressing any other key, ANSWER$ would not contain any letter, and be an empty or null string. If it is a null string, the computer will go back to line 210 and wait for another key to be pressed. If the string is not empty, then it must contain an "N" or a "Y." If it contains a "Y," the computer will go back to line 160 and wait for another cost to be entered.

Line 250 prints once to skip a line, then ends the program.

We can change lines 200 to 210 to accept a default value. Look at the two alternate lines printed below the main listing. Line 200 is changed to include a space and a "Y" after the question mark. This will be printed on the screen. In line 210, the value after SIZE is changed to a −1. Now when the program is run, the "Y" will appear on the screen after the question. If there is another sale, you need only press ENTER. The computer will accept the "Y" that is already on the screen as the entry, unless the N key and ENTER are pressed to indicate "No."

Chapter 9

Storing Related Information

Sometimes the information that the program needs will change every time the routine is used by the computer; however, it is not necessary for the user to provide the information. The program can contain this information within itself. This information is called *data.*

READ/DATA/RESTORE

Program data is stored in one or more program lines. The computer will not use this information until it is told to. The READ command directs the computer to the information in the DATA lines. The computer starts with the first DATA line and uses the first piece of information there, if there is more than one piece of information on the line, it will continue with this data line until all the information is read. Each time the computer uses the READ command it will get the next piece of information. The data can be numbers or letters. Numbers can be read into numeric variables or string variables. Letters can only be read into string variables. The format for the READ command is:

```
30 READ C   or
30 READ C$
```

More than one variable can be used on a program line if your program will be reading several pieces of data at the same time.

```
30 READ C,C$
```

The data is stored on a DATA line. Each piece of data must be separated from the other by a comma. The DATA line can hold a maximum of 31 elements, or 30 commas. Any more, even if the maximum length of the line is not filled, will produce a LINE TOO LONG error.

```
100 DATA red,green,orange,blue,violet,yellow
110 DATA 4,5,6,7,12,145,34
120 DATA name,4,red,3,girl,4,hello,5
```

After the computer has reached the last piece of data, another READ command will cause an error

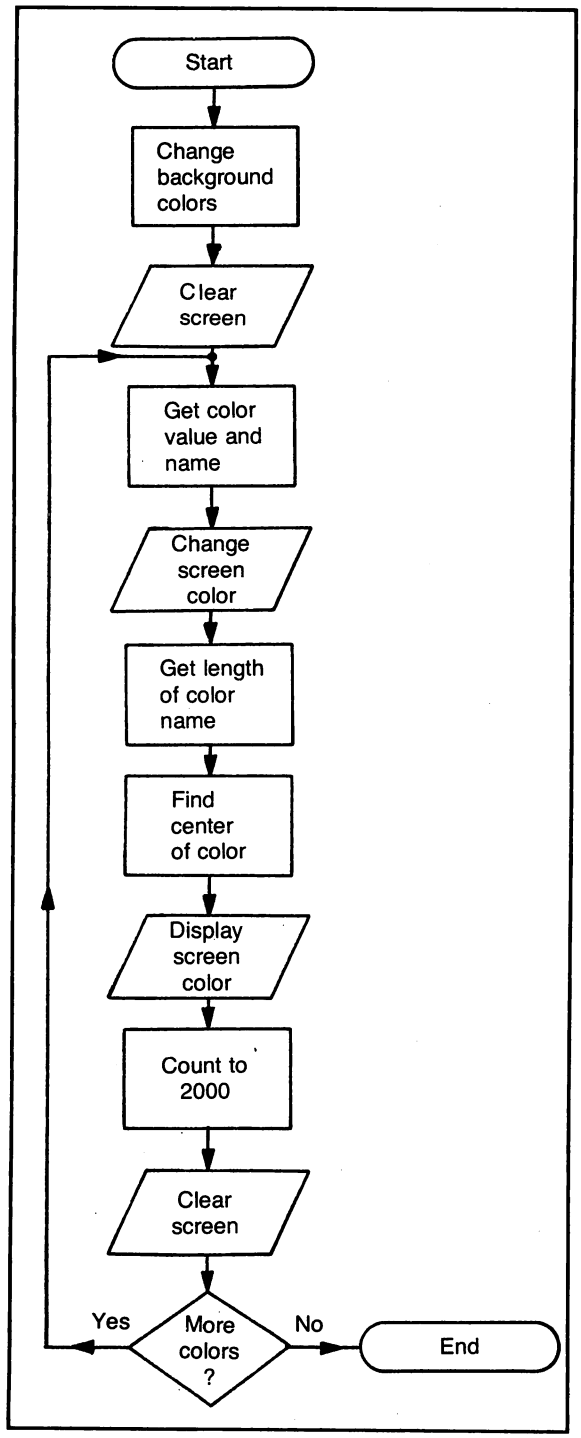


Fig. 9-1. Flowchart for Listing 9-1 Colors.

message to appear on the screen. The data can be reused with the RESTORE command. RESTORE tells the computer to start pointing to the first line or a specific DATA line. This way, the information can be reused. The computer will start from the first line of data if no line number is specified with the RESTORE command.

```
60 RESTORE
```

The computer will begin with a certain line of data if a line number is specified, as shown below.

```
60 RESTORE 110
```

In the program in Listing 9-1 (flowcharted in Fig. 9-1), the information in the DATA lines tells the computer what color the screen should be. The screen color will also be printed. The program uses the RESTORE command without the line number so the program continues the color cycle until the FCTN and 4 keys are pressed.

Listing 9-1

Line 130 changes the colors of the letters from the third set through the ninth set. (The first set is numbered zero.) The I variable represents which set will be changed. The number two changes the character color to black. The 16 changes the background of that letter to white. Now the letters will show up on all the screen background colors.

Line 140 clears the screen.

Line 150 READs the information from the DATA lines (240-250). The number from the DATA line is stored in the CLRCD variable. The color is stored in the string variable CLRNAM$. Since we are reading the number and color as a set, the READ command reads the number and the word with one command followed by both variables. The comma separates the variables on the line.

Line 160 changes the color of the screen to the color value just read. The CALL SCREEN command changes the screen color. This color is determined by the value of the CLRCD variable.

Line 170 finds the length of the word and stores it in

Listing 9-1

```
100 REM LISTING 9-1
110 REM COLORS
120 REM L.M.SCHREIBER FOR TAB BOOKS
130 FOR I=2 TO 8 :: CALL COLOR(I,2,16)::
 NEXT I
140 CALL CLEAR
150 READ CLRCD,CLRNAM$
160 CALL SCREEN(CLRCD)
170 L=LEN(CLRNAM$)
180 CNTR=14-L/2
190 DISPLAY AT(20,CNTR):CLRNAM$
200 FOR DELAY=1 TO 2000 :: NEXT DELAY
210 CALL CLEAR
220 IF CLRCD=16 THEN RESTORE
230 GOTO 150
240 DATA 1,TRANSPARENT,2,BLACK,3,MEDIUM
GREEN,4,LIGHT GREEN,5,DARK BLUE,6,LIGHT
BLUE,7,DARK RED,8,CYAN
250 DATA 9,MEDIUM RED,10,LIGHT RED,11,DA
RK YELLOW,12,LIGHT YELLOW,13,DARK GREEN,
14,MAGENTA,15,GRAY,16,WHITE
```

the L variable. This length will be used to center the word on the screen.

Line 180 divides the length of the word in half, then subtracts that number from 14. The fourteenth position is the center of the screen. By subtracting half the length of the word from the center, we can center the word on the screen.

Line 190 uses the DISPLAY AT command to print the color on the screen. You will notice that the word is printed in black and the background color of the letters is white.

Line 200 is a delay loop. If we did not place a delay loop in the program, the screen would change colors too fast for you to read the color name on the screen.

Line 210 clears the screen again.

Line 220 checks the value of the CLRCD variable. If the value of this variable is 16, the computer uses the RESTORE command to set the pointer back to the beginning of the DATA lines.

Line 230 sends the computer back to line 150. The program will continue to change the screen color and print that color on the screen until the FCTN and 4 keys are pressed.

Lines 240-250 contain the data that the computer uses to change the screen colors and print the colors on the screen. The number indicates the color that the screen will be, and the word is the corresponding color.

WHAT IS AN ARRAY?

An array is a set of locations used for storing and/or retrieving information. Numbers or letters can be stored in an array. Figure 9-2 shows how an array is arranged. This is a one-dimensional array, because it contains only one row. If a teacher wants

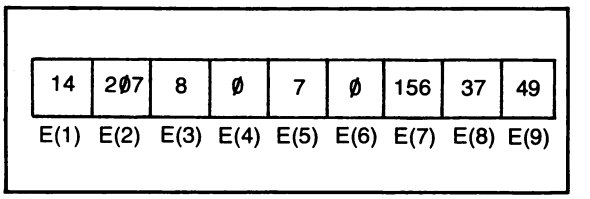


Fig. 9-2. Storing numbers in a numeric array.

to record the grades for her class, and she knows that there are 25 students and each will take eight tests in the quarter, to record this information she would use an array that is 25 rows by eight columns.

In another program the array could hold a predetermined value for plotting points on the screen or determining various statistics. An insurance program could have an array that would contain various ages and the rate of insurance for each age group.

USING ARRAYS

DIM

The DIM or *dimension* command is used to tell the computer how large the array will be. Each location in the array is called an *element*. If you will be using ten or less elements in the array, you do not have to use the DIMension command. TI BASIC uses the default value of ten for arrays. In the program in Listing 9-2 (flowcharted in Fig. 9-3) we will be using only seven elements of the array.

Listing 9-2

Line 130 sets the option base for the program. It is possible to use the zero element of an array. If you do not need to use it, you can tell the computer to start the array with the first element with the OPTION BASE command.

Line 140 tells the computer that the MILES array will only use seven elements.

Line 150 clears the screen.

Line 160 places a message on the fourth row beginning with the first column.

Line 170 begins a FOR . . . NEXT loop. The DAY variable will count to seven, the number of days that the trip was for.

Line 180 asks the user to enter the number of miles driven on a particular day. If you look closely at this line, you will see that the number of the day is stored in the variable DAY. This variable is printed before the question mark and after the word "day." The beep is sounded and the ACCEPT command uses the VALIDATE option. Only a number can be entered. This number will be stored in the MILES array. The value of the

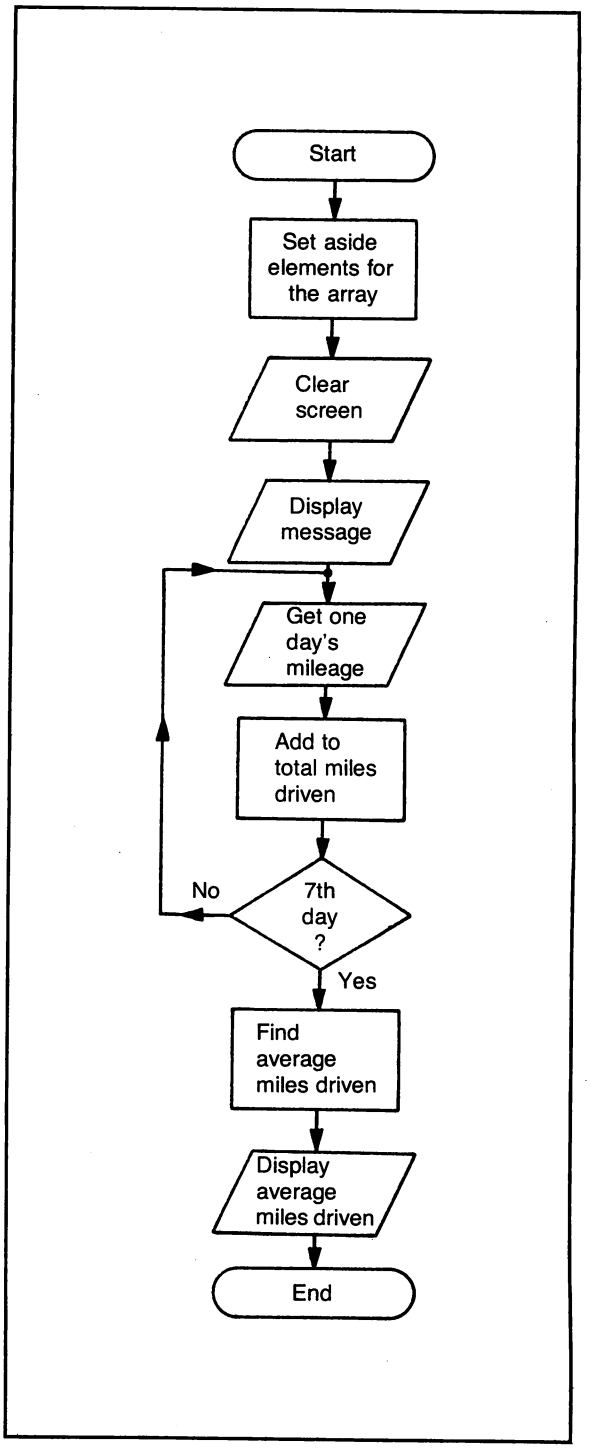


Fig. 9-3. Flowchart for Listing 9-2 Mileage.

Listing 9-2

```
100 REM LISTING 9-2
110 REM MILEAGE
120 REM L.M.SCHREIBER FOR TAB BOOKS
130 OPTION BASE 1
140 DIM MILES(7)! STORAGE SPACE FOR MILE
AGES FOR 7 DAYS
150 CALL CLEAR
160 DISPLAY AT(4,1):"This program will c
alculate the average number of miles dri
ven on a 7 day trip."
170 FOR DAY=1 TO 7
180 DISPLAY AT(12,1):"HOW MANY MILES WER
E DRIVEN  ON DAY" :: DISPLAY AT(13,8):DA
Y;"?" :: ACCEPT AT(13,13)BEEP VALIDATE(N
UMERIC):MILES(DAY)
190 TOTAL=TOTAL+MILES(DAY):: NEXT DAY
200 AVERAGE=INT(TOTAL/7+.5)
210 DISPLAY AT(12,1):TOTAL;"WERE THE TOT
AL MILES" :: DISPLAY AT(13,1):"DRIVEN IN
 THE 7 DAYS."
220 DISPLAY AT(16,1):"THE AVERAGE NUMBER
 OF MILES DRIVEN IN A DAY WAS";AVERAGE;"
."
230 END
```

variable indicates which element of the array will contain the information.

Line 190 contains a running total for the number of miles driven. Each day's mileage is added to the previous total. The program continues until the number of miles for all seven days has been entered.

Line 200 finds the average mileage. The total number of miles are divided by the number of days in the trip (7). Since this number does not have to be a whole number, we need to round it. Add .5 to the average, then take the integer or whole number. For example, 827 divided by 7 equals 118.14; add .5 to this number and it becomes 118.54. The integer is 118, so the average miles driven per day would be 118. However, if 900 miles were driven, then the average number of miles driven per day would be 128.57. Add .5 to this amount, yielding 129.07, and take the integer or whole number of that result, 129.

Line 210 displays the total miles driven.

Line 220 shows the average miles driven per day.

Line 230 ends the program.

The program in Listing 9-2 uses a one-dimensional numeric variable array. The one-dimensional string array works the same way. The next program (Listing 9-3) is a simplified spelling program. (See flowchart in Fig. 9-4.) The words are entered into a one-dimensional string array. These words are then flashed on the screen for the child.

Listing 9-3

Line 130 dimensions the string array WORD$ for 20 elements. A maximum of 20 words can be stored in this program.

Line 140 clears the screen.

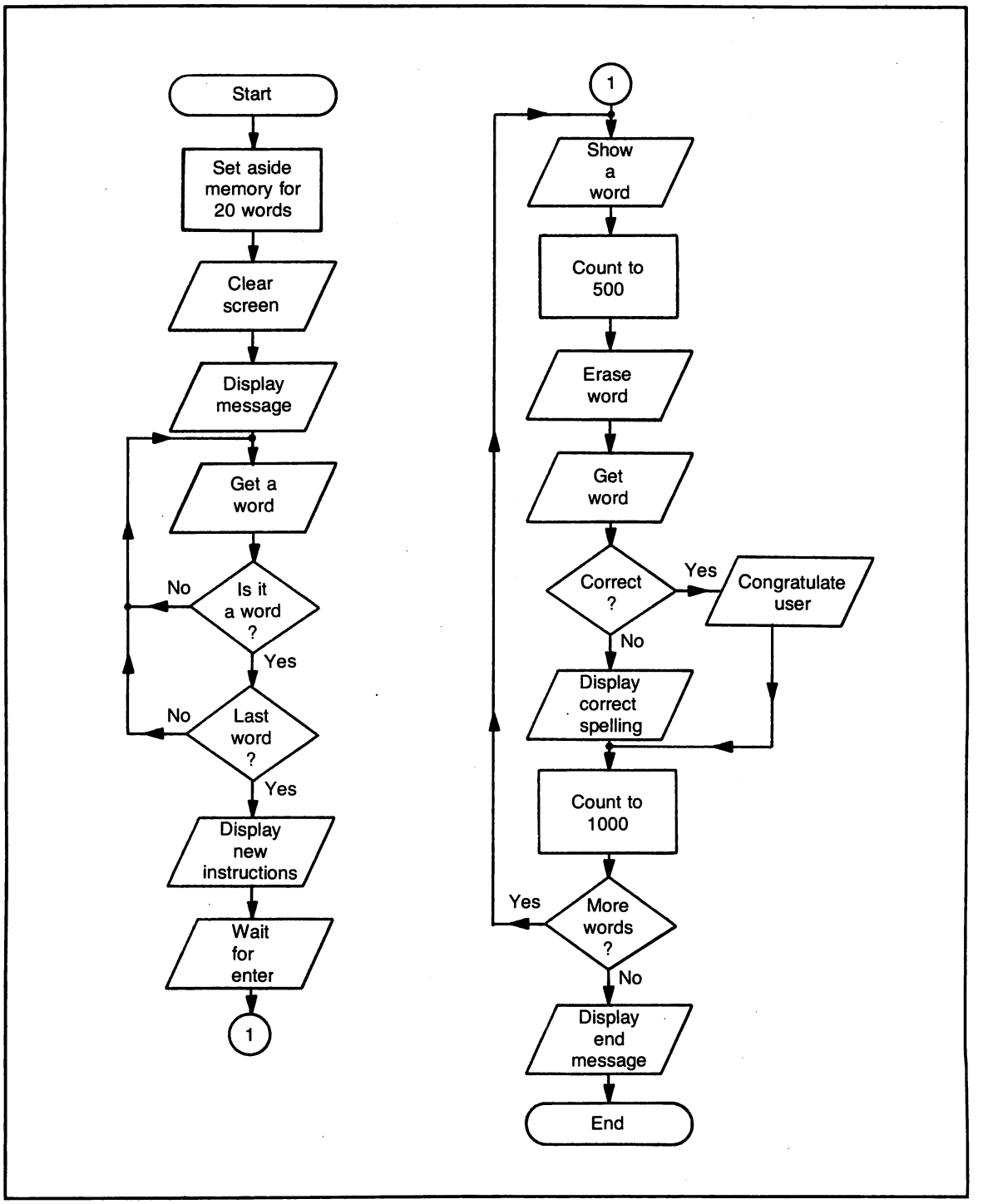


Fig. 9-4. Flowchart for Listing 9-3 Spelling.

Listing 9-3

```
100 REM LISTING 9-3
110 REM SPELLING
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DIM WORD$(20)
140 CALL CLEAR
150 DISPLAY AT(5,5):"This program will a
llow" :: DISPLAY AT(6,1):"you to enter u
p to twenty   spelling words."
160 DISPLAY AT(9,5):"TYPE - xxx when you
 have" :: DISPLAY AT(10,1):"no more word
s to enter and  the last word is not the
    twentieth."
170 FOR COUNT=1 TO 20 :: DISPLAY AT(15,5
):"PLEASE ENTER YOUR WORD"
180 DISPLAY AT(17,5):"#";COUNT :: ACCEPT
 AT(17,10)BEEP SIZE(18):ANSWER$
190 IF ANSWER$="" THEN 180
200 IF ANSWER$="XXX" THEN 240
210 WORD$(COUNT)=ANSWER$
220 NEXT COUNT
230 REM NOW FOR THE TEST
240 DISPLAY AT(1,1)ERASE ALL BEEP:" " ::
 DISPLAY AT(12,1)BEEP:"PRESS 'ENTER' WHE
N YOU ARE  READY TO START"
250 ACCEPT AT(13,16)BEEP:A$
260 FOR TSTCNT=1 TO COUNT-1
270 DISPLAY AT(12,5)ERASE ALL BEEP:WORD$
(TSTCNT)
280 FOR DELAY=1 TO 500 :: NEXT DELAY
290 DISPLAY AT(12,5)BEEP:"ENTER THE WORD
" :: ACCEPT AT(14,7):ANSWER$
300 IF ANSWER$=WORD$(TSTCNT)THEN DISPLAY
 AT(16,7):"VERY GOOD!!!!!" :: GOTO 320
310 DISPLAY AT(16,4):"WRONG - - THE WORD
 WAS" :: DISPLAY AT(18,10):WORD$(TSTCNT)
320 FOR DELAY=1 TO 1000 :: NEXT DELAY
330 NEXT TSTCNT
340 DISPLAY AT(22,1)BEEP:"YOU HAVE FINIS
HED THE LESSON"
```

Lines 150-160 print the instructions on the screen. The DISPLAY AT command is used to print the words exactly where we want them on the screen. Line 170 begins a FOR . . . NEXT loop. This loop will accept up to 20 words. The message PLEASE ENTER YOUR WORD appears on the screen. Line 180 uses the DISPLAY AT command to print "#" and the number of the word being entered.

Because the variable COUNT is printed on the screen, you will know how many words have been entered and which word you are currently entering. The computer will beep and erase 18 spaces on the screen. Up to 18 letters can be entered for each word. The word entered is stored in ANSWER$.

Line 190 checks the contents of ANSWER$. If it is empty, then no word has been entered, and the computer will go back to line 180 to wait for a word.

Line 200 checks to see if the word is XXX. If it is, then the computer knows that you are done with this part of the program. The computer is sent to the next part of the program that tests the user on these words.

Line 210 places the spelling word entered into the correct element of WORD$. The value of the variable COUNT will indicate the next available location in WORD$. The spelling word is placed there.

Line 220 continues the loop. This part of the program or routine will continue until 20 words have been entered, or until the user enters an XXX.

Line 240 begins the test part of the program. The screen is erased and the message is displayed on the screen.

Line 250 waits until the ENTER key has been pressed. This way, the program will not begin until the user is ready.

Line 260 begins another FOR . . . NEXT loop. The computer will count from one until one less than the value of COUNT. We do not want to count up to the value of COUNT because it is one more than the number of words entered.

Line 270 erases the screen and prints a spelling word on the screen. Since the variable TSTCNT will begin with one and end with the last number of the word entered, the computer will begin with the first word that was entered and continue through all the words in WORD$. The variable TSTCNT is the number of the word being displayed.

Line 280 is a delay loop that keeps the word on the screen long enough to be read, but not so long that the user can study it. If you would like the word to be removed from the screen more quickly, change 500 to a smaller number. If you would like to leave the word up longer, increase the value from 500 to a higher number.

Line 290 erases the word and asks the user to enter the word that was just on the screen. The computer will wait until a word is entered. The word is stored in ANSWER$.

Line 300 checks the word contained in ANSWER$ against the word that was on the screen. If both words are the same, the message VERY GOOD will be displayed on the screen and the computer is directed to line 320; it will skip line 310.

Line 310 is used when the word entered is not the same as the word that was flashed on the screen. This line tells the user that the word entered was wrong and displays the correct word on the screen.

Line 320 is another timing loop. This one gives you a chance to read the message.

Line 330 continues the program until all the words have been flashed on the screen.

Line 340 ends the program with a message.

In the previous programs, you used one-dimensional arrays. At other times, you may need a two-dimensional array. You can think of your screen as a two-dimensional array. All the characters are placed in a particular row and column. In a two-dimensional array, you must tell the computer how many rows and columns the array will need (Fig. 9-5).

	1	2
1	red	rojo
2	blue	azul
3	green	verde
4	black	negro
5	white	blanco
6	yellow	amarillo

Fig. 9-5. Storing words in a two-dimension string array.

```
40 DIM A(4,5), D$(8,2)
```

In this example, the numeric array will use four rows and five columns to store numbers. The string array will use eight rows and two columns. The program in Listing 9-4 (Fig. 9-6) uses a two-dimensional string array to store words.

Listing 9-4

Line 130 sets aside the memory needed for the string array. The array called WORD$ will use six rows and two columns.

Line 140 clears the screen.

Line 150 begins a FOR . . . NEXT loop. The variable COUNT will count from one to six.

Line 160 reads two words from the DATA line. The first word is stored in the first column of the array, the second in the second column. The variable COUNT will indicate which element of the array will continue this routine until all six sets of words have been read.

Line 180 begins another FOR . . . NEXT loop.

Line 190 displays the English word on the screen. The word that will be displayed will be determined by the value of the variable COUNT. If the value of COUNT is four, the word "black" will be printed on the screen.

Line 200 prints a message on the screen.

Line 210 tells the computer to sound a beep and wait for an answer. The answer will be stored in the string variable ANSWER$.

Line 220 checks the answer that was entered. If the word entered is not the same as the word stored in the second column of WORD$ array, the program will go back to line 210 for another try. Each set of words will be stored in corresponding parts of the string array. For example, the third row, first column of the array is the word green. The third row, second column of the array is verde, the Spanish word for green. The variable COUNT points to the correct row of the array. This variable does not change its value until the correct Spanish color is entered.

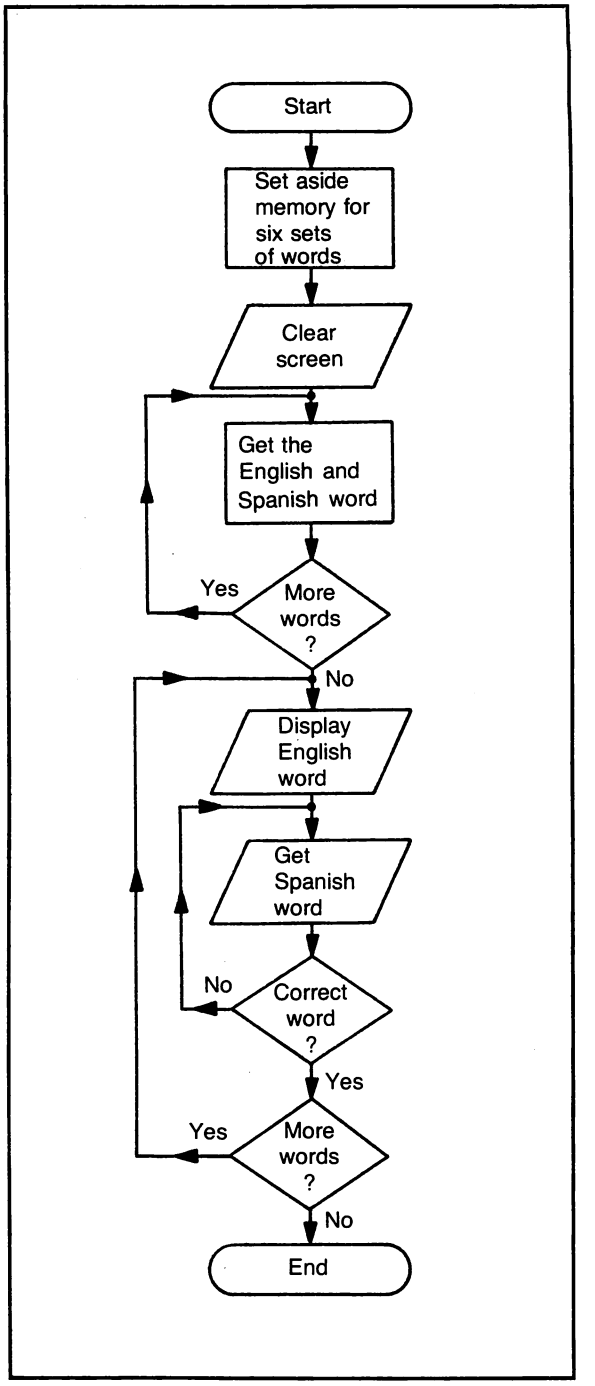


Fig. 9-6. Flowchart for Listing 9-4 Spanish Colors.

Listing 9-4

```
100 REM LISTING 9-4
110 REM SPANISH COLORS
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DIM WORD$(6,2)
140 CALL CLEAR
150 FOR COUNT=1 TO 6
160 READ WORD$(COUNT,1),WORD$(COUNT,2)
170 NEXT COUNT
180 FOR COUNT=1 TO 6
190 DISPLAY AT(12,1):"ENGLISH WORD IS ";
WORD$(COUNT,1)
200 DISPLAY AT(14,1):"SPANISH WORD IS"
210 ACCEPT AT(14,17)BEEP:ANSWER$
220 IF ANSWER$<>WORD$(COUNT,2)THEN 210
230 NEXT COUNT
240 DATA RED,ROJO
250 DATA BLUE,AZUL
260 DATA GREEN,VERDE
270 DATA BLACK,NEGRO
280 DATA WHITE,BLANCO
290 DATA YELLOW,AMARILLO
```

Line 230 the program continues until all six colors have been displayed on the screen.

Lines 240-290 are the DATA lines. These are the words that are stored in the array WORD$. As you can see, each line contains the English and Spanish color.

Chapter 10
Making Decisions in Programs

Programs are not always straightforward calculations of accumulated information. When we figured out the area of a room or moved letters across the screen, the program ran from start to finish without any consideration of the information entered by the user. It processed everything in the order that it was instructed.

Some of the programs that we've entered so far did take into consideration the entries. The Spelling program allowed the user to stop the first routine by entering XXX. When the computer must choose between different program paths, we are talking about *logic* or *decision making statements.* The computer must decide which path to take. This decision is determined by information that has been entered or calculated in the previous part of the program.

DECISION-MAKING STATEMENTS

IF . . . THEN

The very simplest decision making statement is an IF . . . THEN statement. IF the first part of the statement is true, THEN the program continues with the second part of the statement. These statements are often used after an input statement to check the answer entered for erroneous answers. Other times it is used after a computation to decide on the path the computer must take in the program. Listing 10-1 (flowcharted in Fig. 10-1) shows an example of IF . . . THEN statements. Use the keys that have the arrows on the side of them. Do not use the "FCTN" key. These keys will control the ball on the screen. The IF . . . THEN statements keep the ball on the screen.

Listing 10-1

Line 130 sets the ROW variable to 12 and the COL variable to 14, the center location of the screen. The twelfth ROW is halfway down and the fourteenth COL is halfway across.

Line 150 prints a warning message on the screen saying that the ALPHA LOCK key must be pressed down or this program will not operate correctly.

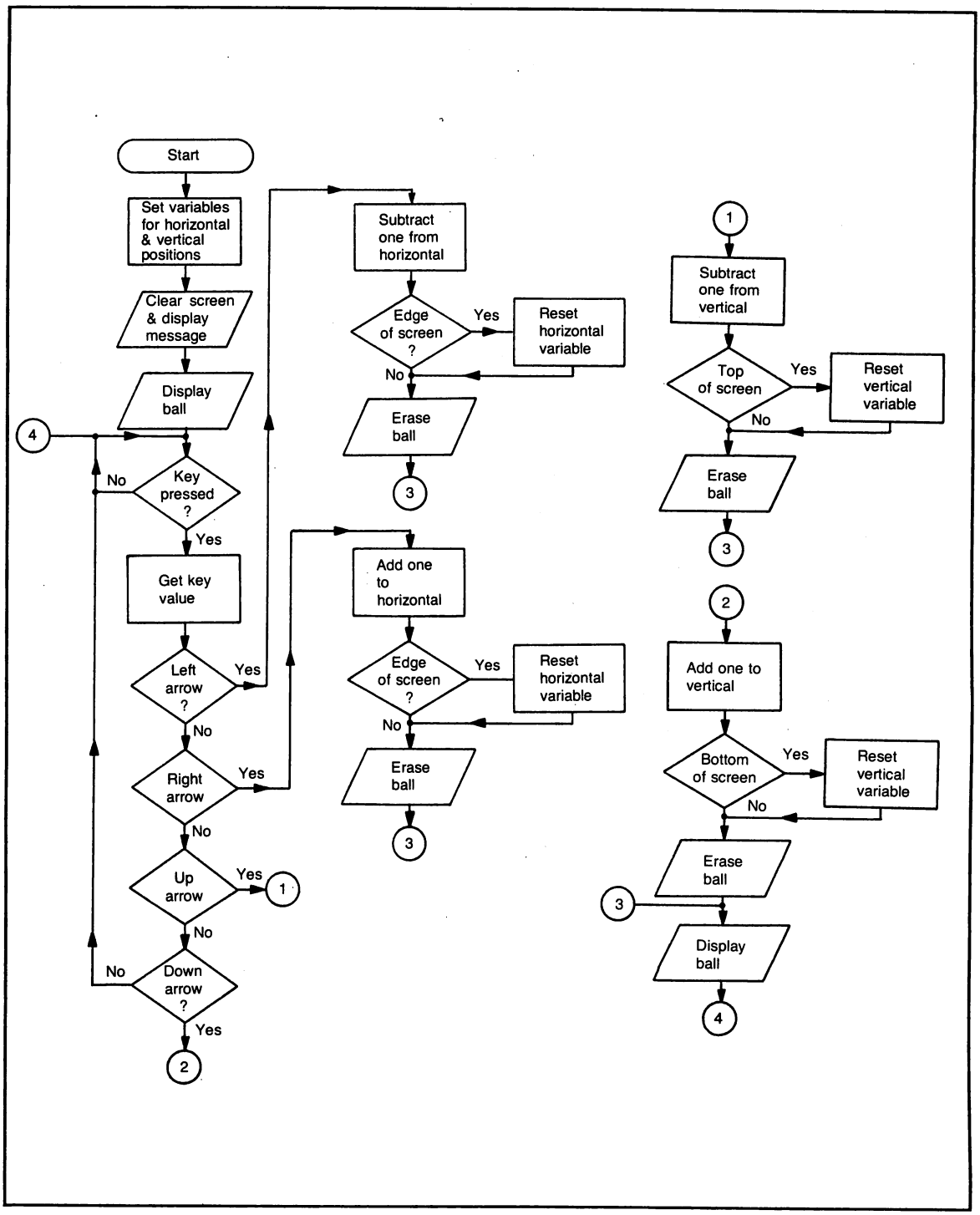


Fig. 10-1. Flowchart for Listing 10-1 Ball.

Listing 10-1

```
100 REM LISTING 10-1
110 REM BALL
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 ROW=12 :: COL=14 ! SET UP FOR CENTER
 SCREEN
140 REM CLEAN UP SCREEN AND DISPLAY A ME
SSAGE
150 CALL CLEAR :: PRINT "MAKE SURE ALPHA
 LOCK KEY":"IS IN DOWN (LOCK) POSITION"
160 FOR DELAY=1 TO 2000 :: NEXT DELAY
170 REM CLEAR MESSAGE OFF SCREEN
180 CALL CLEAR
190 DISPLAY AT(ROW,COL):"O" ! PLACE BALL
 ON SCREEN
200 REM TEST FOR ANY KEY PRESSED
210 CALL KEY(0,KEY,STATUS)
220 REM IF NOT, GO BACK & CHECK IT AGAIN
230 IF STATUS=0 THEN 210
240 REM A KEY WAS PRESSED - SO TEST TO S
EE WHICH ONE
250 IF KEY=83 THEN 300 ! MOVE THE BALL T
O THE LEFT
260 IF KEY=68 THEN 330 ! MOVE BALL TO TH
E RIGHT
270 IF KEY=69 THEN 360 ! MOVE BALL UP
280 IF KEY=88 THEN 390 ! MOVE BALL DOWN
290 GOTO 210 ! NOT A VALID ENTRY
300 COL=COL-1 :: IF COL<1 THEN COL=1 ! C
HECK FOR EDGE OF SCREEN
310 DISPLAY AT(ROW,COL+1):" " ! ERASE OL
D BALL
320 GOTO 420 ! GO TO SECTION TO DISPLAY
BALL IN NEW POSITION
330 COL=COL+1 :: IF COL=29 THEN COL=28 !
 DON'T GO PAST RIGHT EDGE
340 DISPLAY AT(ROW,COL-1):" " ! ERASE OL
D BALL
350 GOTO 420
360 ROW=ROW-1 :: IF ROW=0 THEN ROW=1 ! T
HERE IS NO ROW '0'
370 DISPLAY AT(ROW+1,COL):" " ! ERASE OL
D BALL
380 GOTO 420
```

```
390 ROW=ROW+1 :: IF ROW=25 THEN ROW=24 !
 DON'T GO PAST BOTTOM ROW
400 DISPLAY AT(ROW-1,COL):" " ! ERASE OL
D BALL
410 REM DISPLAY BALL IN NEW POSITION
420 DISPLAY AT(ROW,COL):"O"
430 REM GO BACK AND GET NEXT KEY
440 GOTO 210
```

Line 160 is a delay loop to give you time to read the message.

Line 180 clears the message.

Line 190 places the ball on the screen. In this program the uppercase "O" will represent the ball.

Line 210 uses a CALL command. This command will check to see what key, if any, has been pressed. If a key has been pressed, the STATUS variable will be set and the KEY variable will contain the value of the key that has been pressed.

Line 230 uses an IF . . . THEN statement. IF the value of STATUS is a zero, then a key was not pressed and the computer will go to line 210. If the value of STATUS is other than zero, then the computer will not continue with statement, but go on to the next line of the program.

Line 250 tests the value of KEY. If its value is 83, then the S key was pressed and the computer is directed to line 300.

Line 260 checks the value of KEY for a 68. If it is, then the D key was pressed and the computer will go to line 330.

Line 270 looks for 69. This is the value KEY will be if the E was pressed. The computer will be directed to line 360.

Line 280 checks for the value of 88. This is the value of the X key. When this key is pressed, the computer will go to line 390.

Line 290 directs the computer to line 210. The computer will reach this line if KEY does not equal any of the previous values.

Line 300 moves the ball to the left edge of the screen. One is subtracted from the COL value. This variable is then tested for a value that is less than one. If the COL variable becomes a zero, it is printed off the screen, so, when COL is less than one, it is reset to the value one.

Line 310 erases the ball from its present position on the screen. One is added to the value of COL because the old position is one more than the new position. Printing a space will erase the old ball.

Line 320 sends the computer to line 420 where the ball will be reprinted on the screen.

Line 330 adds one to the value of COL. This will move the ball to the right. Again the value of COL is checked for the edge of the screen. This time, if the value of COL reaches 29, it will be reset to 28 so that the ball will not be printed off screen.

Line 340 erases the ball from the previous position.

Line 350 sends the computer to line 420 to reprint the ball.

Line 360 adjusts the ROW variable. By subtracting one from its value, we can move the ball up on the screen. The value of ROW is checked for a zero. If it is zero, it is reset to one.

Line 370 erases the ball from the screen. This time we are adding one to ROW since that is the variable that we just subtracted one from.

Line 380 directs the computer to line 420 to print the ball in the new position.

Line 390 adds one to the value of ROW. The ball can now be printed one row lower on the screen. The value of ROW is tested for 25. If it reaches 25, it is reset to 24. This keeps the ball on the screen.

Line 400 erases the ball that is currently on the screen.

Line 420 prints the ball on the screen. The values of ROW or COL have been adjusted for the new position on the screen. If the ball has reached any

edge of the screen, it will not move since that variable has been reset to the edge position.

Line 440 sends the computer back to line 210 where it waits for another key to be pressed.

USING IF ... THEN TO EXIT A LOOP

Another use for IF. . .THEN statements is to exit a loop. An example of when you would use IF . . . THEN as an exit is as follows: You are getting information from the user, but you do not determine ahead of time the exact number of entries the user will enter. In the Spelling program, the user could enter up to 20 words, but it is possible to enter only one word. The code XXX signifies the end on the word list. The program checks each entry to see if it is the final entry. When the code is entered, the program leaves the routine it is in and directs the computer to the spelling routine.

In the last program, when an arrow key was pressed, the program exited the routine that checked the value of the key pressed. If an unacceptable key was pressed, the program directed the computer to wait for another key.

MORE DECISION-MAKING STATEMENTS

ELSE

Sometimes you may want the computer to do one of two things depending on what the circumstance is. Instead of using two IF . . . THEN statements, one for each possibility, the ELSE can be used to tell the computer which direction it should take.

For example, ELSE can be used in a program that shows the tax imposed on income earnings above a certain level. Incomes below $25,000 are taxed at 15 percent and incomes above $25,000 are taxed at 17 percent.

```
215  IF INCOME<25000 THEN TAX=.15 ELSE
     TAX=.17
```

The same idea can be used to direct the computer to different parts of a program.

```
400  IF C=3 THEN 450 ELSE GOTO 500
```

GOTO

The GOTO command can be used in a loop. The number following the GOTO command is the line number that the computer will process next. The line number must be an actual number used in the program. If you try to GOTO a line that does not exist, an error will occur.

ON . . . GOTO

In some programs you may have several routines that can be used, but they will not be used at the same time, or in the same order. When we want the computer to go to a routine only when certain conditions are met, we are using selective branching. One example is a program containing several games or learning modules. When the program is run, the screen contains a menu from which the user can choose a program or unit. The program in Listing 10-2 (see flowchart in Fig. 10-2) uses selective branching for a three unit program on states.

Listing 10-2

Line 130 sets aside two strings. One string is used for the names of the 50 states. The second string is used for the second part of the answer whether it be the capital, the state abbreviation, or the state flower.

Line 150 contains a RESTORE command. The first time that this program is run, the computer will be pointing to the first DATA line (line 420). When the program repeats itself, the computer will be pointing to a different line number. By restoring the pointer to this line, the computer will be able to read the names of the states into the STATE$ array no matter where it was pointing. The FOR . . . NEXT loop reads the names of the states and places them into each element of the string array.

Lines 160-170 clear the screen and place the menu on the screen. You are given three units to choose from.

Line 180 uses the ACCEPT AT command to get the unit number. The VALIDATE option checks the entry. If it is not a 1, 2, or 3, the number will not be accepted. The number entered is stored in UNIT.

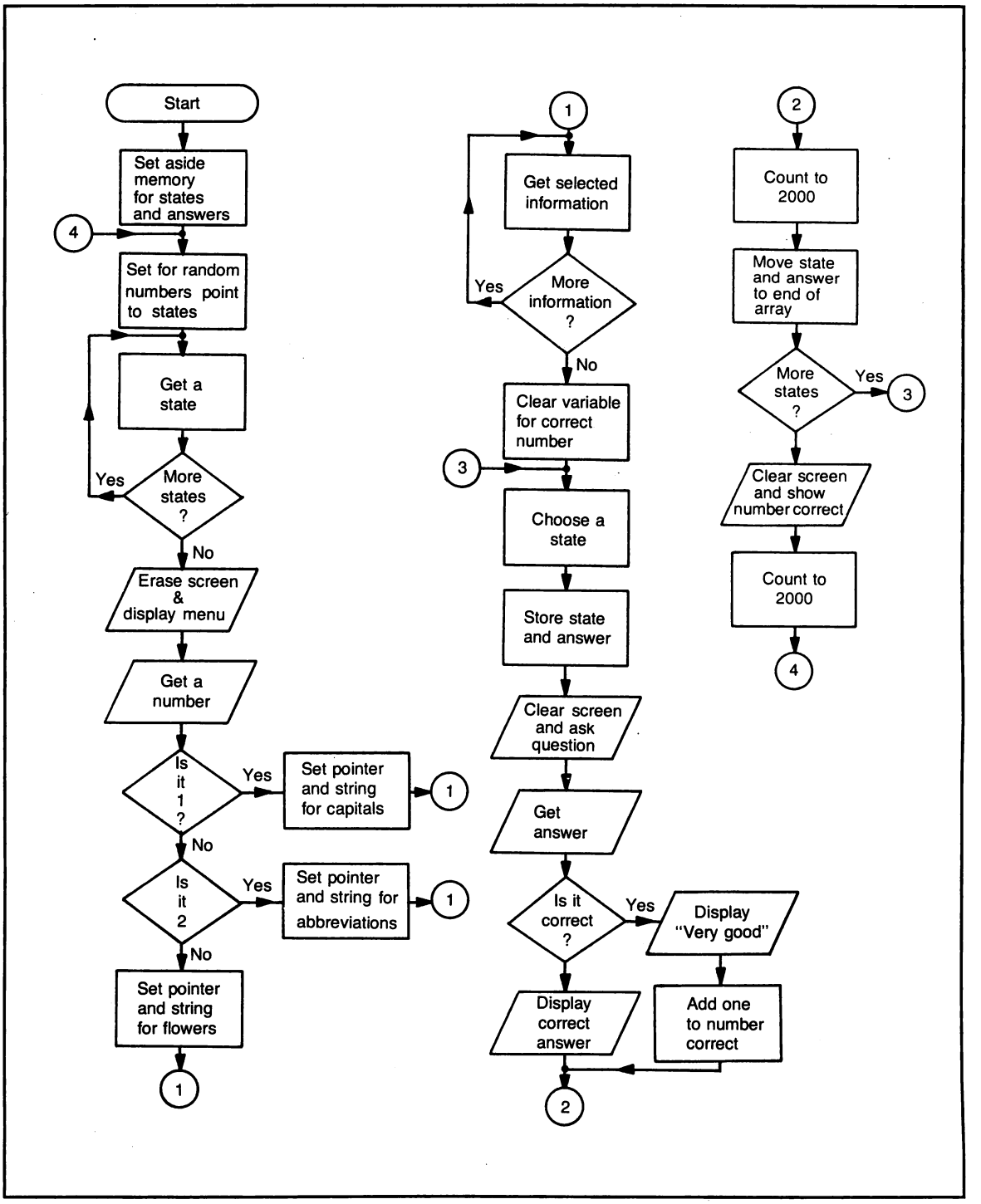


Fig. 10-2. Flowchart for Listing 10-2 Selective Branching.

Line 190 contains the ON . . . GOTO command. The value of UNIT determines which routine in the program the computer will go to. There are three line numbers following GOTO. Each of these line numbers is the first line of that routine.

Line 210 begins the states and capitals routine. This part of the program displays a state on the screen and asks you to enter the correct capital. The RESTORE command sets the pointer to line 470, the first line of data for the capitals. The CATEGORY$ variable is set to "capital." This information is used in the next part of the program.

Line 220 sends the computer to line 280. The question and answer part of the program begins at this program statement.

Line 240 contains another RESTORE command. The computer is directed to this line when unit two is selected. Line 520 is the first line of data of the two-letter abbreviations of the states. CATEGORY$ is set to "abbreviation."

Line 250 sends the computer to line 280 for the questions and answers.

Line 270 restores the pointer to line 550 which contains the list of flowers. Again CATEGORY$ is set to the name of this unit, "flower." There is no GOTO line for this part of the program since line 280 is the next line number.

Line 280 begins the question and answer part of the program. The information from the previous lines is used in this part of the program. The computer knows where to begin reading the information because the RESTORE command set the pointer to the correct DATA lines. Now the computer can READ these lines and place that information into the string array ANSWER$.

Line 290 sets the variable CORRECT to 0. This variable will count how many questions were answered correctly. The FOR . . . NEXT loop makes sure that every state is placed on the screen. The RND chooses a number from one to the value of the COUNT variable. The first time that this routine is used, the COUNT variable is 50. The second time 49, then 48, 47, and so on. The value of S will determine which state will be placed on the screen.

Line 300 takes the state stored at S location in the STATE$ array and places it in TESTSTATE$. This is the state that will be printed on the screen. The corresponding correct answer is stored in CORRECT$. As long as the data lines are entered correctly, STATE$ and ANSWER$ should match at the same location. For example, the first element of STATE$, STATE$(1), is Nebraska. ANSWER$(1) should be NE if the second unit, state abbreviations, has been chosen. TESTSTATE$ is now the state and CORRECT$ is the corresponding correct answer.

Line 310 erases the screen, then prints the question on the screen. CATEGORY$ will print the type of answer that the computer is looking for—the capital, the abbreviation, or the flower—depending on which unit number was entered. TESTSTATE$ prints the state in question.

Line 320 prints the question mark, then beeps. The VALIDATE option limits the entry to letters only. The size limits the number of letters entered to 18. The answer is stored in TRY$.

Line 330 checks the answer that was entered against the correct answer. If it is correct, VERY GOOD is displayed on the screen and the CORRECT variable has one added to it. The computer is directed to line 350.

Line 340 will be used only if the wrong answer is entered. The computer will print the correct answer on the screen.

Line 350 is a timing loop. This gives you time to read the message on the screen.

Line 360 takes the state that is in the last place and places it in the position of the state that has been used. It moves the answer the same way. If this is the first time the routine is used, it will move the state from the 50th position. On the second time the forty-ninth, then the forty-eighth, forty-seventh and so on. This way, every state will be used in a random order.

Line 380 continues the loop. This loop will continue until COUNT is less than one and all the states have been displayed on the screen.

Line 390 erases the screen, then prints your score on the screen.

Line 400 is another timing loop to give you a chance

Listing 10-2

```
100 REM LISTING 10-2
110 REM SELECTIVE BRANCHING
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DIM STATE$(50),ANSWER$(50):: RANDOMI
ZE
140 REM GET STATES INTO STORAGE AREA
150 RESTORE 420 :: FOR COUNT=1 TO 50 ::
READ STATE$(COUNT):: NEXT COUNT
160 DISPLAY AT(6,1)ERASE ALL:"PLEASE CHO
OSE A UNIT (1-3)" :: DISPLAY AT(9,4):"1)
 STATES & CAPITALS"
170 DISPLAY AT(11,4):"2) STATE ABBREVIAT
IONS" :: DISPLAY AT(13,4):"3) STATE FLOW
ERS"
180 ACCEPT AT(6,28)BEEP VALIDATE("123")S
IZE(1):UNIT
190 ON UNIT GOTO 210,240,270
200 REM STATES & CAPITALS
210 RESTORE 470 :: CATEGORY$="CAPITAL"
220 GOTO 280
230 REM STATE ABBREVIATIONS
240 RESTORE 520 :: CATEGORY$="ABBREVIATI
ON"
250 GOTO 280
260 REM STATE FLOWERS
270 RESTORE 550 :: CATEGORY$="FLOWER"
280 FOR COUNT=1 TO 50 :: READ ANSWER$(CO
UNT):: NEXT COUNT
290 CORRECT=0 :: FOR COUNT=50 TO 1 STEP
-1 :: S=INT(RND*COUNT)+1
300 TESTSTATE$=STATE$(S):: CORRECT$=ANSW
ER$(S)
310 DISPLAY AT(8,5)ERASE ALL:"What is th
e ";CATEGORY$ :: DISPLAY AT(10,8):"of ";
TESTSTATE$
320 DISPLAY AT(12,8):"?" :: ACCEPT AT(12
,10)BEEP VALIDATE(UALPHA)SIZE(18):TRY$
330 IF TRY$=CORRECT$ THEN DISPLAY AT(14,
10):"VERY GOOD" :: CORRECT=CORRECT+1 ::
GOTO 350
340 DISPLAY AT(14,10):"No, it's " :: DIS
PLAY AT(16,11):CORRECT$
350 FOR DELAY=1 TO 2000 :: NEXT DELAY
360 STATE$(S)=STATE$(COUNT):: STATE$(COU
```

```
NT)=TESTSTATE$
370 ANSWER$(S)=ANSWER$(COUNT):: ANSWER$(
COUNT)=CORRECT$
380 NEXT COUNT
390 DISPLAY AT(12,7)ERASE ALL:"YOU GOT";
CORRECT;"CORRECT."
400 FOR DELAY=1 TO 2000 :: NEXT DELAY
410 GOTO 150
420 DATA NEBRASKA,SOUTH DAKOTA,NORTH DAK
OTA,MINNESOTA,KANSAS,IOWA,MISSOURI,TEXAS
,OKLAHOMA,ARKANSAS,ALABAMA,MISSISSIPPI,L
OUISIANA,TENNESSEE
430 DATA NEW MEXICO,ARIZONA,UTAH,IDAHO,C
OLORADO,MONTANA,WYOMING,NEVADA,WASHINGTO
N,HAWAII,OREGON,CALIFORNIA,ALASKA,MAINE,
VERMONT
440 DATA KENTUCKY,RHODE ISLAND,NEW HAMPS
HIRE,MASSACHUSETTS,CONNECTICUT,DELAWARE,
NEW YORK,MARYLAND,NEW JERSEY,PENNSYLVANI
A,WEST VIRGINIA
450 DATA FLORIDA,NORTH CAROLINA,VIRGINIA
,SOUTH CAROLINA,GEORGIA,MICHIGAN,WISCONS
IN,ILLINOIS,INDIANA,OHIO
460 REM STATE CAPITALS
470 DATA LINCOLN,PIERRE,BISMARK,ST. PAUL
,TOPEKA,DES MOINES,JEFFERSON CITY,AUSTIN
,OKLAHOMA CITY,LITTLE ROCK,MONTGOMERY,JA
CKSON,BATON ROUGE
480 DATA NASHVILLE,SANTA FE,PHOENIX,SALT
 LAKE CITY,BOISE,DENVER,HELENA,CHEYENNE,
CARSON CITY,OLYMPIA,HONOLULU,SALEM,SACRA
MENTO,JUNEAU
490 DATA AUGUSTA,MONTPELIER,FRANKFORT,PR
OVIDENCE,CONCORD,BOSTON,HARTFORD,DOVER,A
LBANY,ANNAPOLIS,TRENTON,HARRISBURG,CHARL
ESTON,TALLAHASSEE
500 DATA RALEIGH,RICHMOND,COLUMBIA,ATLAN
TA,LANSING,MADISON,SPRINGFIELD,INDIANAPO
LIS,COLUMBUS
510 REM STATE ABBREVIATIONS
520 DATA NE,SD,ND,MN,KS,IA,MO,TX,OK,AR,A
L,MS,LA,TN,NM,AZ,UT,ID,CO,MT,WY,NV,WA,HI
,OR,CA,AK,ME,VT,KY,RI
```

```
530 DATA NH,MA,CT,DE,NY,MD,NJ,PA,WV,FL,N
C,VA,SC,GA,MI,WI,IL,IN,OH
540 REM STATE FLOWERS
550 DATA GOLDENROD,PASQUEFLOWER,WILD PRA
IRIE ROSE,LADY SLIPPER,SUNFLOWER,WILD RO
SE,HAWTHORN,BLUEBONNET,MISTLETOE,APPLE B
LOSSOM,CAMELIA
560 DATA MAGNOLIA,MAGNOLIA,IRIS,YUCCA FL
OWER,SAGUARO,SAGO LILY,SYRINGE,COLUMBINE
,BITTERROOT,INDIAN PAINTBRUSH,SAGEBRUSH,
RHODODENDRON
570 DATA HIBISCUS,OREGON GRAPE,GOLDEN PO
PPY,FORGET-ME-NOT,PINE CONE,RED CLOVER,G
OLDENROD,VIOLET,PURPLE LILAC,MAYFLOWER,M
OUNTAIN LAUREL
580 DATA PEACH BLOSSOM,ROSE,BLACK-EYED S
USAN,VIOLET,MOUNTAIN LAUREL,RHODODENDRON
,ORANGE BLOSSOM,FLOWERING DOGWOOD,AMERIC
AN DOGWOOD
590 DATA CAROLINA JESSAMINE,CHEROKEE ROS
E,APPLE BLOSSOM,VIOLET,VIOLET,PEONY,SCAR
LET CARNATION
```

to read your score.

Line 410 sends the computer back to line 150. The states will be read back into the array and the menu appears on the screen for another choice.

Lines 420-590 contain the data for this program. This is divided into four parts—the states, their capitals, the two-letter abbreviations, and the state flowers. Do not try to place more two-letter abbreviations on the DATA lines than are there. TI BASIC only allows 30 commas on one DATA line. You will get a line too long error if you try to enter more than 30 commas.

Chapter 11

Repeating Part of the Program

You will often find parts of your program repeating themselves. Typing in the same instructions over and over again is tiring for you and a waste of memory for the computer. Bytes disappear very quickly even in the most memory efficient program.

One way to conserve memory is to place the instruction or set of instructions the computer will be repeating in a *loop*. A loop tells the computer to repeat a certain set of instructions any number of times. In the past few chapters you have used loops for timing routines and input. Loops kept the size of the program reasonable.

USES FOR LOOPS

The computer can process information with remarkable speed. If the computer is also asked to print information on the screen as it processes it, chances are the computer's speed will be too fast for you to read the information. Sometimes just listing a program is too fast!

If you are printing instructions on the screen for the user or presenting a problem for the user to read, you will need to slow the computer down so that the user can read it before it disappears. A *timing loop* was used for this purpose in the Colors program and the Spelling program. Timing loops tell the computer to stay at a particular place in the program and do nothing but count from one number to another. The numbers are not displayed on the screen but serve to slow down the computer to allow the user to read the information on the screen.

Another loop was used in the Spelling program when the user was asked to enter the words. Without it, the program would have to contain 20 input commands, 20 prompt lines, and 20 decision lines. A needless waste of memory! A series of inputs can usually be obtained most efficiently by using a loop.

Beware of looping to infinity! When you construct a loop, you must design an exit from the loop, or you may wait for the computer to complete a calculation, read information, or time an activity, only to discover (after pressing the FCTN and 4 keys, of course) that the computer hasn't passed

line 30! A loop with no exit is called an endless loop; it is useful in demonstration programs, where you want the same program to be repeated all day, or at the end of a program that you want to end without the prompt appearing on the screen. The only way to exit an endless loop is by pressing the FCTN and 4 keys.

FOR . . . NEXT LOOPS

A FOR . . . NEXT loop repeats a set of program instructions a given number of times.

```
20  FOR T=1 TO 100
30  NEXT T
```

This loop starts by setting the T variable to 1. The second command is NEXT T. The program tells the computer to start with the number one, then add one to the value of T and return to the FOR statement. It continues to go back and forth between the FOR and NEXT until T is equal to 100. When the T variable equals 101, it has exceeded the second value and goes on to the program line following the NEXT command.

If we had other program lines between lines 20 and 30, the computer would execute any and all of the commands between the FOR and the NEXT 100 times.

In the above example we started with the T

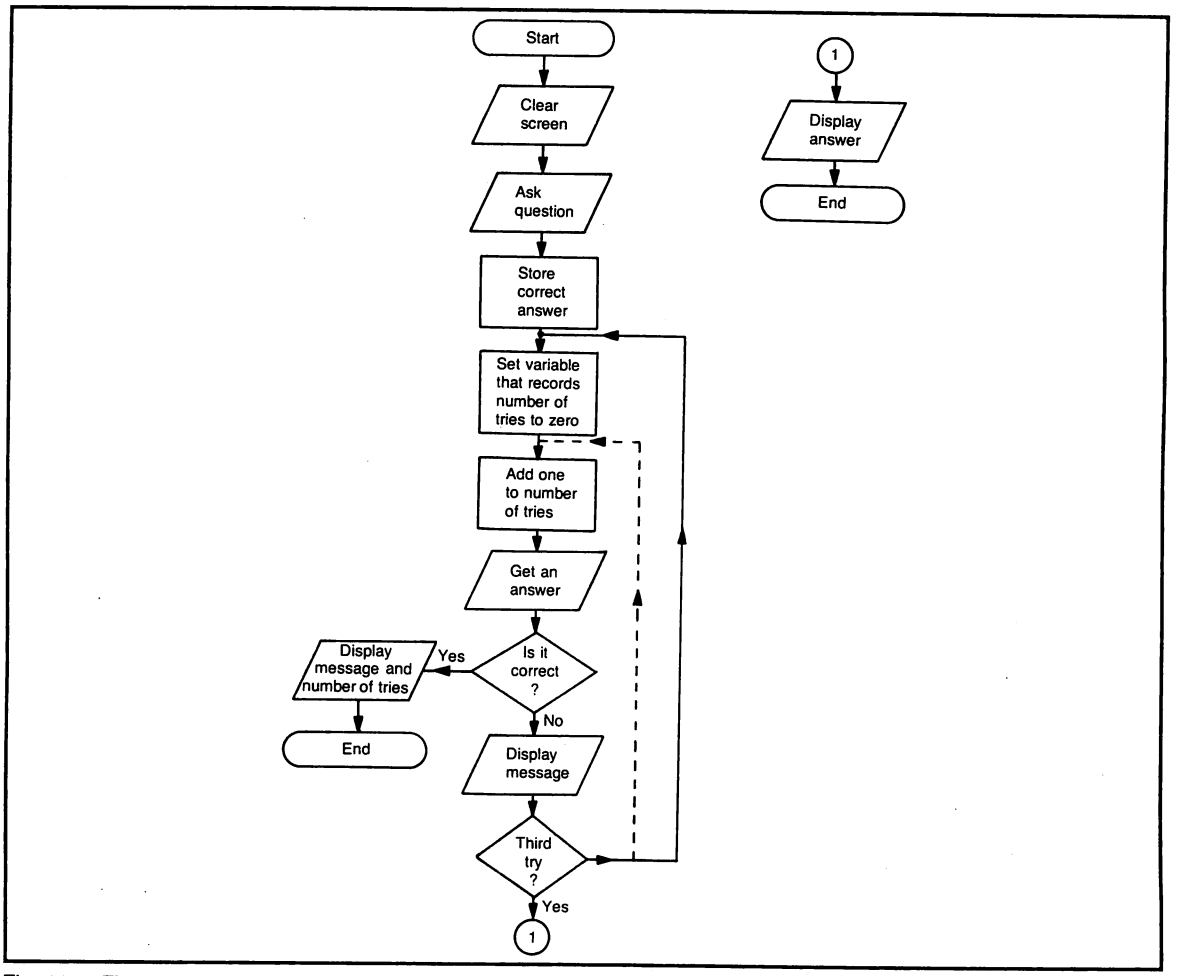


Fig. 11-1. Flowchart for Listings 11-1 and 11-1B, Answer Version 1 and Answer Version 2.

Listing 11-1A

```
100 REM LISTING 11-1A
110 REM ANSWER VERSION 1
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR
140 DISPLAY AT(12,1):"WHAT IS THE CAPITA
L OF       MONTANA ? "
150 ANSWER$="HELENA"
160 FOR COUNT=1 TO 3
170 TRY=0 ! NUMBER OF GUESSES
180 TRY=TRY+1 :: ACCEPT AT(13,12)BEEP:GU
ESS$
190 IF GUESS$=ANSWER$ THEN 240
200 IF COUNT<3 THEN DISPLAY AT(18,10):"T
RY AGAIN"
210 FOR DELAY=1 TO 500 :: NEXT DELAY ::
DISPLAY AT(18,1):" "
220 NEXT COUNT :: DISPLAY AT(18,1):"THE
CAPITAL OF MONTANA IS    ";ANSWER$
230 END
240 DISPLAY AT(18,1):"VERY GOOD !  YOU G
OT IT IN  ";TRY;"TRIES."
```

variable equal to one and ended with it equal to 100. Any variable and any starting and ending numbers can be used in your programs.

One common error when using the FOR . . . NEXT loop is setting a variable to zero within the loop instead of before the computer starts the loop. An example of this would be a program that gives the user three tries to answer a problem. The variable that counts the number of wrong answers must be cleared before each question. If this variable is cleared within the loop, the computer will never know when the three tries are up. Listing 11-1B demonstrates the correct use of FOR . . . NEXT loops. Listing 11-1A shows the same programs with a variable cleared inside the loop that should be cleared before the loop. The flowchart in Fig. 11-1 shows both possible loops. The dotted line points out the correct way to set up the loop.

Listing 11-1A

Line 130 clears the screen.

Line 140 prints the question on the screen on the 12th line and first column.

Line 150 sets the string variable ANSWER$ to Helena.

Line 160 begins the FOR . . . NEXT loop. The COUNT variable will begin with one and continue until it reaches three.

Line 170 sets the number of guesses to zero.

Line 180 counts which guess this is by adding one to the TRY variable. The computer beeps and waits for an answer. The answer will be stored in GUESS$.

Line 190 checks the entry against the correct answer. If the answer is correct, the computer will be directed to line 240.

Line 200 checks to see if this is the last try. If the COUNT variable is less than three, the computer will display "TRY AGAIN" on the screen.

Line 210 is a delay loop. Then the message is erased.

Line 220 sends the computer back to the line where

Listing 11-1B

```
100 REM LISTING 11-1B
110 REM ANSWER VERSION 2
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR
140 DISPLAY AT(12,1):"WHAT IS THE CAPITA
L OF       MONTANA ? "
150 ANSWER$="HELENA"
160 TRY=0 ! NUMBER OF GUESSES
170 FOR COUNT=1 TO 3
180 TRY=TRY+1 :: ACCEPT AT(13,12)BEEP:GU
ESS$
190 IF GUESS$=ANSWER$ THEN 240
200 IF COUNT<3 THEN DISPLAY AT(18,10):"T
RY AGAIN"
210 FOR DELAY=1 TO 500 :: NEXT DELAY ::
DISPLAY AT(18,1):" "
220 NEXT COUNT :: DISPLAY AT(18,1):"THE
CAPITAL OF MONTANA IS    ";ANSWER$
230 END
240 DISPLAY AT(18,1):"VERY GOOD !  YOU G
OT IT IN  ";TRY;"TRIES."
```

the FOR . . . NEXT loop began for another try. When the value of COUNT exceeds three, the answer will be displayed.

Line 230 ends the program so that the congratulatory message is not displayed.

Line 240 congratulated the player for guessing the correct answer.

If you try this program the way it is written, you will find that you will always get the answer in one try whether it took you only one try or not. The reason is that the TRY variable is cleared each time the computer executes the FOR . . . NEXT loop. The result is always:

VERY GOOD! YOU GOT IT IN 1 TRIES

Listing 11-1B corrects this situation by exchanging lines 160 and 170. Now the variable is cleared only before the loop is executed. The program will tell you the correct number of tries that it took before the correct answer was entered.

FOR . . . NEXT loops can also be used within each other. This is called *nesting*. An example of nested loops is shown in the program in Listing 11-2. The flowchart of the program (Fig. 11-2) shows the proper structure of nested loops. Note that the inner loop is completed before the outer loop can go on the next value. The inner loop is also completed each time the outer loop is executed. If you do not nest loops properly, you can get error messages, cause the program to crash, or get incorrect answers.

STEPPING

A FOR . . . NEXT loop does not have to add one to the variable every time it completes the loop. You can have the variable incremented by any amount by adding STEP to the command, as shown below.

```
40  FOR Z=10 TO 100 STEP 5
50  . . .
60  NEXT Z
```

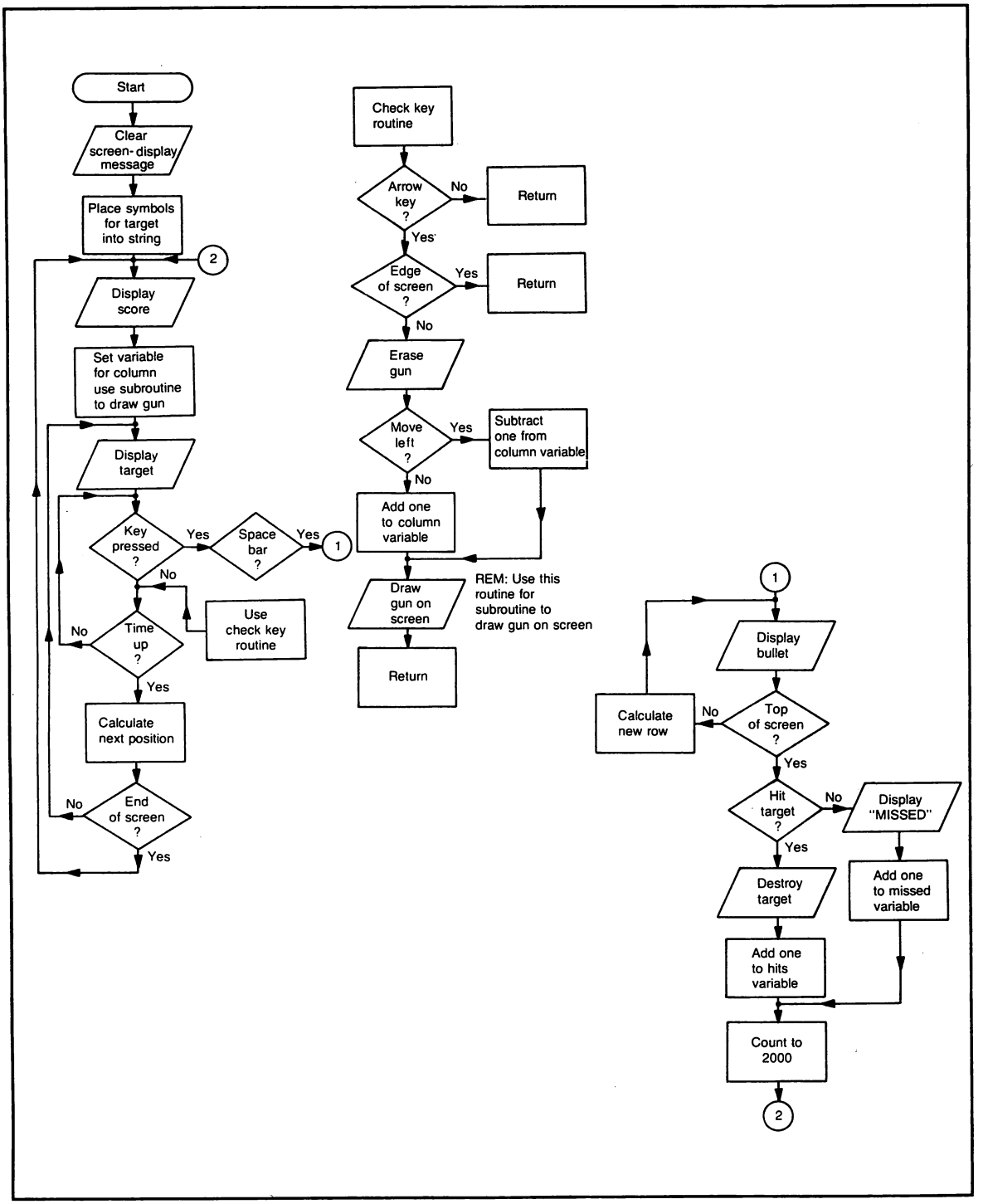


Fig. 11-2. Flowchart for Listing 11-2 Target.

In this program the Z variable will be equal to 10 the first time the computer executes line 40. When it comes to line 60, five will be added to the variable, making it 15. The program will continue with the computer adding five to the value of Z until Z is greater than 100.

If you want the computer to count backwards, use a negative number after the STEP option, as shown below.

```
50  FOR G=150 TO 50 STEP -5
```

Here, the computer will set the G variable to 150 the first time it executes the line. The second time G will be 145; then 140, 130, and so on. When G is less than 50, the computer will continue with the next line of the program.

Listing 11-2 contains examples of FOR . . . NEXT loops.

Listing 11-2

Lines 130-150 clears the screen and prints a short introductory message. This message will remain on the screen until the ERROR key is pressed.

Line 160 sets the TARGET$ variable to an asterisk, two slashes, and a hyphen. This is our target.

Line 170 places the score at the top of the screen. Both the hits and misses will be displayed.

Line 180 sets the COL variable to 13. This is the column that the gun will be printed at. The computer will go to line 340 to print the gun on the screen.

Line 190 is the first loop of the two nested FOR . . . NEXT loops. The POSITION variable will indicate which column the target will be printed at. It will begin with column 25 and work its way from right to left across the screen. Because the column numbers decrease as we travel across the screen, this loop will count backwards.

Line 200 prints the target on the screen based on its new column value.

Line 210 begins the second FOR . . . NEXT loop.

Listing 11-2

```
100 REM LISTING 11-2
110 REM TARGET
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: PRINT "TARGET SHOOT is
 a rifle game": :"PRESS the RIGHT ARROW
to    move the rifle to the right."
140 PRINT :"PRESS the LEFT ARROW to move
 the rifle to the left.": :"PRESS the UP
 ARROW to shoot."
150 PRINT : : :"          GOOD LUCK !!": :
 :"PRESS ENTER TO CONTINUE";:: INPUT A$
160 TARGET$="*//-"
170 DISPLAY AT(1,2)ERASE ALL:"HITS ";HIT
S :: DISPLAY AT(1,15):"MISSED ";MISSES
180 COL=13 :: GOSUB 340
190 FOR POSITION=25 TO 1 STEP -1 ! TARGE
T WILL MOVE FROM RIGHT TO LEFT
200 DISPLAY AT(5,POSITION):TARGET$ ! DRA
W THE TARGET ON THE SCREEN
210 FOR TIME=1 TO 25 :: CALL KEY(1,KEY,S
TATUS)
```

```
220 IF STATUS=1 THEN IF KEY=5 THEN 350 E
LSE GOSUB 260
230 NEXT TIME
240 NEXT POSITION
250 GOTO 170
260 IF KEY=2 OR KEY=3 THEN 280
270 RETURN
280 IF COL<3 AND KEY=2 THEN RETURN
290 IF COL>25 AND KEY=3 THEN RETURN
300 FOR ROW=21 TO 24 :: DISPLAY AT(ROW,C
OL):"    " :: NEXT ROW ! CLEAR OLD GUN
310 IF KEY=2 THEN COL=COL-1 :: GOTO 330
320 COL=COL+1
330 IF COL<2 OR COL>26 THEN RETURN
340 FOR ROW=21 TO 24 :: DISPLAY AT(ROW,C
OL):"| |" :: NEXT ROW :: RETURN
350 FOR BULLETROW=23 TO 5 STEP -1
360 DISPLAY AT(BULLETROW+1,COL+1)SIZE(1)
:" " :: DISPLAY AT(BULLETROW,COL+1)SIZE(
1):"*"
370 NEXT BULLETROW
380 IF COL+1>=POSITION AND COL+1<=POSITI
ON+3 THEN DISPLAY AT(5,POSITION):"::::"
:: HITS=HITS+1 :: GOTO 400
390 DISPLAY AT(3,POSITION-2):"MISSED" ::
 MISSES=MISSES+1
400 FOR DELAY=1 TO 200 :: NEXT DELAY ::
GOTO 170
```

The KEY routine is called to see if a key has been pressed.

Line 220 checks the value of the STATUS variable. If it is a one, then a key has been pressed. The next part of this IF . . . THEN statement is another IF . . . THEN. If the value of key is five, then the program will direct the computer to line 350; however, if the value is anything else, the computer will use the subroutine that begins with line 260.

Line 230 continues the loop until the value of TIME exceeds 25. This loop is nested within the POSITION loop.

Line 240 continues the loop that moves the target across the screen. This loop will continue until the value of POSITION is less than one.

Line 250 sends the computer back to line 170 and repeats the entire program.

Line 260 is the routine that the computer is directed to from line 220. The value of the KEY variable is checked for a two or three. If it contains either of these values, the program will continue at line 280.

Line 270 sends the computer back to line 230 because the value of KEY was neither two nor three.

Line 280 checks the value of the COL and KEY variables. If COL is less than three, then the gun cannot move to the left any further. If the value of KEY is two, then the user wants to move the gun to the left. It cannot move, so the computer returns to the line that sent it to this routine.

Line 290 checks the value of COL and KEY again. This time it is checking to see if the gun is as far to the right as it can go and the user wants to move it further. If both of these conditions are true, that is, the value of COL is greater than 25 and the value of KEY is three, the computer will return to the line that sent it to this routine.

Line 300 removes the gun from the screen. Now it can draw the gun in its new position.

Line 310 looks at the value of KEY to see if the COL variable must have one added to it or subtracted from it. If the value of KEY is two then one will be subtracted from the value of COL and the program will direct the computer to line 30.

Line 320 adds on to the value of COL. We know that the value of KEY can only be a two or three since we tested it in line 260 for those two values. Since it was tested for the value of two in line 310 and failed the test, we know that its value can only be a three. Therefore, we do not have to test it for its value again.

Line 330 checks the value of COL to make sure that it is not less than two or greater than 26. If it is the computer will leave this routine and return to the line that sent it.

Line 340 draws the gun on the screen. The value of the ROW will change, but the value of COL remains the same. The computer returns to the line that sent it.

Line 350 begins another FOR . . . NEXT loop. This time we place the bullet on the screen. It will start at the bottom of the screen and continue up to the line the target is on. The rows on the screen decrease as we travel up the screen, so this loop counts backwards.

Line 360 erases the last bullet. The first time through this loop there will be no bullet to erase. The COL variable is increased by one. This variable holds the position of the gun on the screen. The bullet is in the next column. The BULLETROW variable is the row that the bullet is on. We start with 23, which is one row up from the bottom of the screen. After the bullet is erased, we draw the bullet in the new position on the screen.

Line 370 continues the loop until the bullet reaches the row that the target is on.

Line 380 compares the position of the target with the position of the bullet. The POSITION variable indicates the position of the target. The target is four characters long. If the position of the bullet, which is one more than the COL variable, is equal to the position of the target, or is not greater than the last character of the target (POSITION+3), we have hit the target. The target will be replaced with four colons. The HITS variable will be increased by one and the program will go on to line 400.

Line 390 will be executed if the bullet is not within the range specified for the target. MISSED will be printed above the target and the MISSED variable will be increased by one.

Line 400 contains another timing loop. The computer is then directed back to line 170.

Listing 11-3 is a routine for shuffling cards, flowcharted in Fig. 11-3. You may want to use it in any program where you will be using information, numbers, or words, and do not want to repeat the same routine twice.

This method replaces the item chosen with the last item in the array; takes the last one and places it in the location chosen, and then decreases the number of locations that the computer can choose from. The locations that the information is moved to cannot be disturbed because the computer will not be allowed to choose from those locations.

Listing 11-3

Line 130 sets aside 52 locations for the cards.

Line 140 places the numbers and letters of the cards into the string. The Ace is the lowest card and the King is the highest.

Line 150 is a FOR . . . NEXT loop. The computer will count from 1 to 52. Each time it will place the value of X in the CARDNO array. Every element of CARDNO will contain a number from 1 to 52.

Line 160 contains the RANDOMIZE command. Without this program line the computer would shuffle the cards the same way every time the program was run.

Line 170 begins the FOR . . . NEXT loop that will

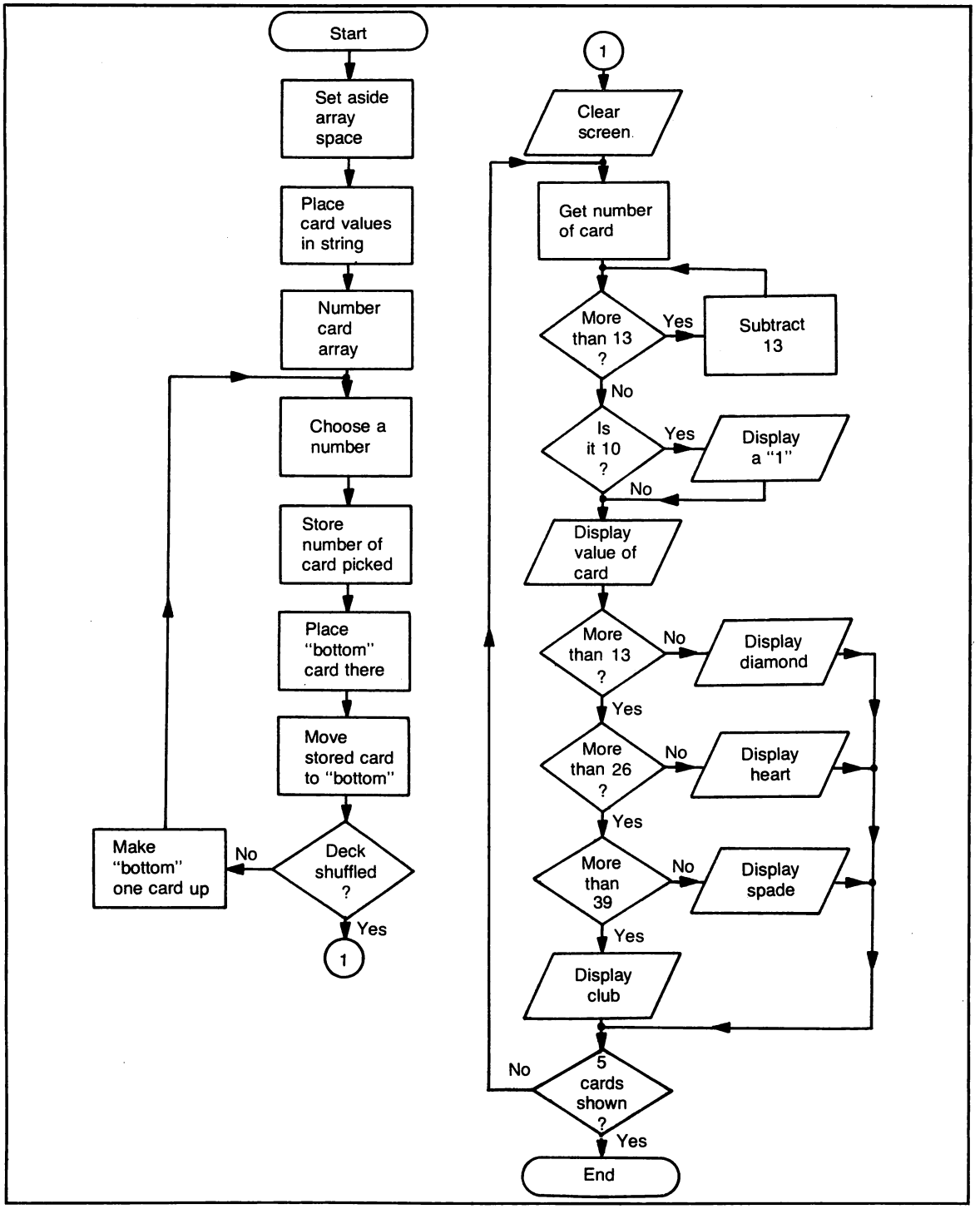


Fig. 11-3. Flowchart for Listing 11-3 Shuffle.

Listing 11-3

```
100 REM LISTING 11-3
110 REM SHUFFLE
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DIM CARDNO(52)
140 CARDNAME$="A234567890JQK"
150 FOR COUNT=1 TO 52 :: CARDNO(COUNT)=C
OUNT :: NEXT COUNT ! NUMBER THE CARDS
160 RANDOMIZE ! GET NEW SERIES OF RANDOM
 NUMBERS
170 FOR COUNT=52 TO 1 STEP -1
180 SHUFFLE=INT(RND*COUNT)+1 ! CHOOSE A
CARD FROM 1 TO THE NUMBER LEFT IN THE DE
CK
190 REM 'TEMP' IS A TEMPORARY STORAGE VA
RIABLE
200 TEMP=CARDNO(SHUFFLE)! SWAP THE CARDS
 - STORE THE CARD AT LOCATION 'SHUFFLE'
IN THE VARIABLE 'TEMP'
210 CARDNO(SHUFFLE)=CARDNO(COUNT)! TRANS
FER THE CARD FROM LOCATION 'COUNT' TO 'S
HUFFLE'
220 CARDNO(COUNT)=TEMP ! PLACE THE CARD
REMOVED FROM 'SHUFFLE' INTO LOCATION 'CO
UNT'
230 NEXT COUNT
240 REM PRINT THE FIRST 5 CARDS IN THE S
HUFFLED DECK
250 CALL CLEAR
260 FOR COUNT=1 TO 5
270 REM GET CARD NUMBER INTO TWO SEPARAT
E VARIABLES
280 CRNTNO=CARDNO(COUNT):: CARDPOS=CARDN
O(COUNT)
290 REM ADJUST 'CARDPOS' UNTIL IT POINTS
 AT CORRECT LOCATION IN 'CARDNAME$'
300 IF CARDPOS>13 THEN CARDPOS=CARDPOS-1
3 :: GOTO 300
310 IF CARDPOS=10 THEN PRINT "1";! NECES
SARY AS 'CARDPOS' CAN ONLY POINT AT THE
'0'
320 REM PRINT THE ONE CHARACTER CODE PER
 'CARDPOS' OF 'CARDNAME$' (THE VALUE OF
THE CARD)
330 PRINT SEG$(CARDNAME$,CARDPOS,1);" ";
```

```
340 IF CRNTNO>13 THEN 360 ! IT'S NOT A D
IAMOND
350 PRINT "DIAMOND" :: GOTO 420
360 IF CRNTNO>26 THEN 380 ! IT'S NOT A H
EART
370 PRINT "HEART" :: GOTO 420
380 IF CRNTNO>39 THEN 410 ! IT'S NOT A S
PADE
390 PRINT "SPADE" :: GOTO 420
400 REM IT IS A CLUB
410 PRINT "CLUB"
420 NEXT COUNT
430 END
```

shuffle the cards. We want to start with a full deck, so make COUNT equal to 52 and count backwards.

Line 180 picks one of the cards. The computer will be allowed to choose one number from one to the value of COUNT. The first time that the computer executes this line it can choose any of the 52 cards. The second time 51, the third time 50, and so on.

Line 200 places the card that the computer picked in a temporary location. We will call this location TEMP.

Line 210 takes the card at the bottom of the pile and places it in the location that we just removed a card from. COUNT will always represent the bottom of the pile. The first time the card is taken from location 52, the second time 51, the third time 50, and so on. Since COUNT is always decreasing, we will not take a card twice.

Line 220 transfers the card from the temporary location (TEMP) to the bottom of the pile. Again, COUNT will be decreasing, so the number placed in the last element of the array cannot be chosen or replaced once COUNT has decreased.

Line 230 continues the loop until all the cards have been moved.

Line 250 clears the screen.

Line 260 begins another FOR . . . NEXT loop. This time we want only the first five cards in the array printed on the screen.

Line 280 takes the value of the COUNT element of the array and places it into CRNTNO and CARDPOS variables. CARDPOS will be the letter or number of the card.

Line 300 checks the value of CARDPOS. If it is greater than 13, CARDPOS will be decremented by 13 until it is less than or equal to 13. The computer subtracts 13 from the value of CARDPOS because there are 13 cards in each suit. Since this value can be any value from 1 to 52, the computer needs to subtract 13 from it until the number is equal to or less than 13. Then it will know what the value of the card is.

Line 310 checks the value of CARDPOS again. If it is 10, the computer will print a one on the screen, and use a semicolon to hold the cursor there for the rest of the card. CARDNAME$ can only contain one letter or number for each card—the ten card is the exception to the number/suit pattern.

Line 330 prints the number or letter of the card on the screen. The SEG$ command takes a letter or number from CARDNAME$ variable. The value of CARDPOS is a number from one to 13. This position in CARDNAME$ contains the corresponding number or letter of the card. Since we only want one number or letter, the number "one" tells the computer to take only one character from this string at the CARDPOS position; this character will be printed on the screen. Use the semicolon to keep the cursor on that line.

Line 340 checks the value of CRNTNO. This variable will indicate what suit should be printed on the screen. The cards contain four suits, with thirteen cards in each suit. If the value of CRNTNO is greater than 13, then the card will not be a diamond and the computer will go on to line 360.

Line 350 prints the suit of the card. In this case, a diamond.

Lines 360-410 continue checking the value of CRNTNO for the correct suit. When it finds the suit of the card, it prints that suit on the screen.

Line 420 continues the FOR . . . NEXT loop until all five cards have been printed on the screen.

Line 160 in this program contains a RANDOMIZE command. We will discuss this command in greater detail in a later chapter. You may want to delete this line from your program, then run it several times and note the cards that come up on the screen. Each time you run the program without the RANDOMIZE command, your cards should be the same. (Talk about a stacked deck!) With the RANDOMIZE command, the cards will be different each time the program is run.

This shuffle routine can be changed to shuffle the array no matter how many elements it has. It can also be used in routines that will shuffle words to be displayed, as in a Spelling program.

Chapter 12

Reusing Part of the Program

In Chapter 10 we discussed routines that were used selectively by the program. These routines could be used more than once, but only after the entire routine was completed and the program had displayed the menu. What happens if we have a routine used by several parts of the same program? If this routine will be used by the main part of the program, and we expect to come back to the same part of the program that we left, we will need some way to keep track of where we are and where we are going. You could use a series of IF . . . THEN statements or list the routine in the program wherever you need it, but each of these methods wastes time and memory.

USING A SUBROUTINE

GOSUB . . . RETURN

The best way to handle a routine that you will call often is to replace the multiple copies of the routine with one *subroutine*. A subroutine is part of the program that can be used at any time in the program. When the computer finishes with the subroutine it returns to the part of the program that it came from. One example is a timing loop. You will often use the same timing loop at several points in your program. You could write the timing routine once and use it as a subroutine from *any* point in your program. If, for example, you had a routine that played a certain melody, and you wanted to use that routine several times in your program, it would save a good amount of work if you made that music routine a subroutine.

When the computer finds a GOSUB command, it remembers the line number by placing it in an area of memory called a stack. It then goes to the line number that appears after GOSUB. It executes the lines in the subroutine until it encounters a RETURN command. This command tells the computer to go back to the line it came from and continue with the program. GOSUB is used in Listing 12-1 (flowcharted in Fig. 12-1).

Listing 12-1

Line 140 uses the OPTION BASE command. This

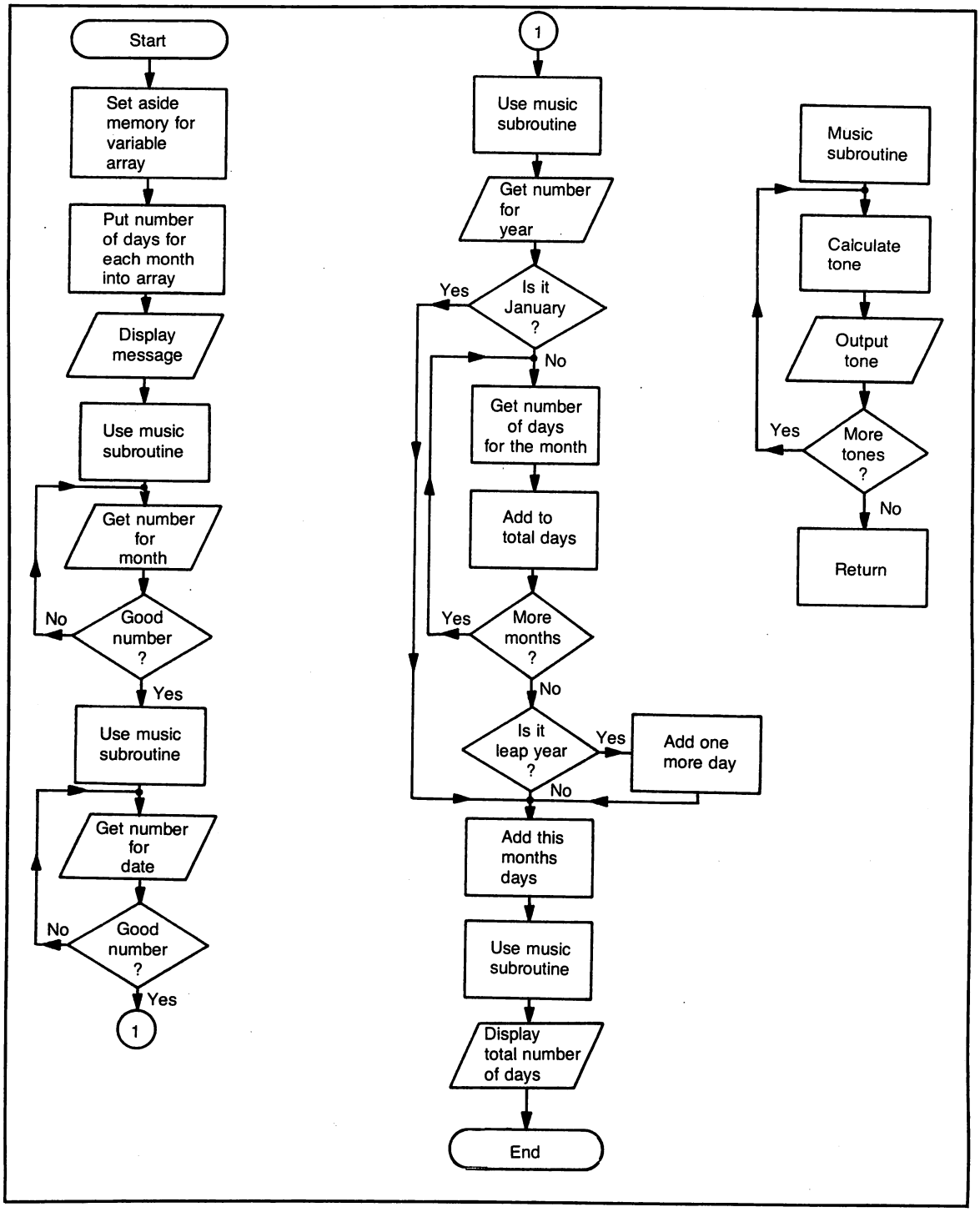


Fig. 12-1. Flowchart for Listing 12-1 Days.

Listing 12-1

```
100 REM LISTING 12-1
110 REM DAYS
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 REM SET UP ARRAY FOR DAYS IN EACH MO
NTH - DON'T USE ARRAY ELEMENT 0
140 OPTION BASE 1 :: DIM DAYS(12)
150 FOR COUNT=1 TO 12 :: READ DAYS(COUNT
)::: NEXT COUNT
160 DATA 31,28,31,30,31,30,31,31,30,31,3
0,31
170 DISPLAY AT(4,1)ERASE ALL:"THIS IS A
DEMONSTRATION OF ASUBROUTINE"
180 REM USE SUBROUTINE TO PLAY A SCALE &
 THEN CONTINUE PROGRAM AT NEXT LINE
190 GOSUB 370
200 DISPLAY AT(8,1):"PLEASE ENTER TODAY'
S DATE"
210 DISPLAY AT(10,1):"MONTH (NUMBER ONLY
) ?" :: ACCEPT AT(10,24)BEEP VALIDATE(DI
GIT)SIZE(2):MONTH :: IF MONTH<1 OR MONTH
>12 THEN 210
220 REM USE SUBROUTINE AGAIN CONTINUING
PROGRAM AT NEXT LINE
230 GOSUB 370
240 DISPLAY AT(12,1):"DATE (1-31) ?" ::
ACCEPT AT(12,24)BEEP VALIDATE(DIGIT)SIZE
(2):DATE
250 IF DATE>DAYS(MONTH)THEN 240
260 REM USE SUBROUTINE AGAIN THEN CONTIN
UE PROGRAM ON SAME LINE BUT NEXT STATEME
NT
270 GOSUB 370 :: DISPLAY AT(14,1):"YEAR"
 :: ACCEPT AT(14,22)BEEP VALIDATE(DIGIT)
SIZE(4):YEAR
280 IF MONTH=1 THEN 330
290 FOR PAST=1 TO MONTH-1
300 TOTALDAYS=TOTALDAYS+DAYS(PAST)
310 NEXT PAST
320 IF YEAR/4=INT(YEAR/4)THEN IF MONTH>2
 THEN TOTALDAYS=TOTALDAYS+1 ! CHECK FOR
LEAP YEAR
330 TOTALDAYS=TOTALDAYS+DATE
340 GOSUB 370 :: DISPLAY AT(20,1):"TODAY
 IS THE";TOTALDAYS;"th DAY OF" :: DISPLA
```

```
Y AT(21,1):"THE YEAR."
350 END ! DON'T LET THE PROGRAM RUN INTO
 THE SUBROUTINE
360 REM MUSIC SUBROUTINE
370 FOR TONE=300 TO 400 STEP 10
380 CALL SOUND(250,TONE,0)
390 NEXT TONE
400 RETURN ! GO BACK TO THE LINE YOU CAM
E FROM
```

statement will use only the elements one to 12 in the DAYS array. Without this command you could use the zero element of the DAYS array. This line also sets aside 12 locations for use by the DAYS variable.

Line 150 contains a FOR . . . NEXT loop to READ the number of days in each month into the DAYS array.

Line 160 contains the number of days for each month of the year.

Line 170 erases the screen and prints a short message.

Line 190 contains the GOSUB command. The computer is directed to line 370. It will know that it should come back to this point after it completes the subroutine.

Line 200 prints a message on the screen. You are asked to enter today's date.

Line 210 asks you to enter the number of the current month and stores it in the MONTH variable. The VALIDATE option accepts only number keys and the SIZE option limits the number of digits that can be entered to two. The number entered is checked to make sure that it is a valid month. If it isn't, the line will repeat until a valid number is entered.

Line 230 directs the computer to the same subroutine that it used before. The computer will return to this point in the program.

Line 240 prints the next question on the screen. This time you are asked to enter today's date. Again the VALIDATE option is used to accept only numbers, and the SIZE option limits the number of digits to two. This entry will be stored in the DATE variable.

Line 250 checks the date entered against the number of days that the month could legally have.

Line 270 uses the same subroutine that has been used in previous program statements. This time, when the computer returns from the subroutine, it will not go on to the next program, but will continue with this line and ask you to enter the current year. The SIZE option in this line is set to four so that the entire number of the year can be entered (1984 instead of 84). The year will be stored in the YEAR variable.

Line 280 checks the value of the MONTH variable. If the value is a one, then the program will go on to line 330 since it does not have to add any days to the number of days in January.

Lines 290-310 total the number of days of each month that has passed. The program stops one month before the month that has been entered because all the days of the current month have not passed.

Line 320 checks for a leap year by dividing the year by four; if the result is a whole number, it is a leap year. If it is a leap year, the statement checks to see if we have passed February. If the month entered is greater than two, the computer will add one to the total number of days.

Line 330 adds the value of DATE to the total number of days that have passed.

Line 340 uses the subroutine in line 370, then prints which day of the year today is.

Line 350 ends the program. It is very important to

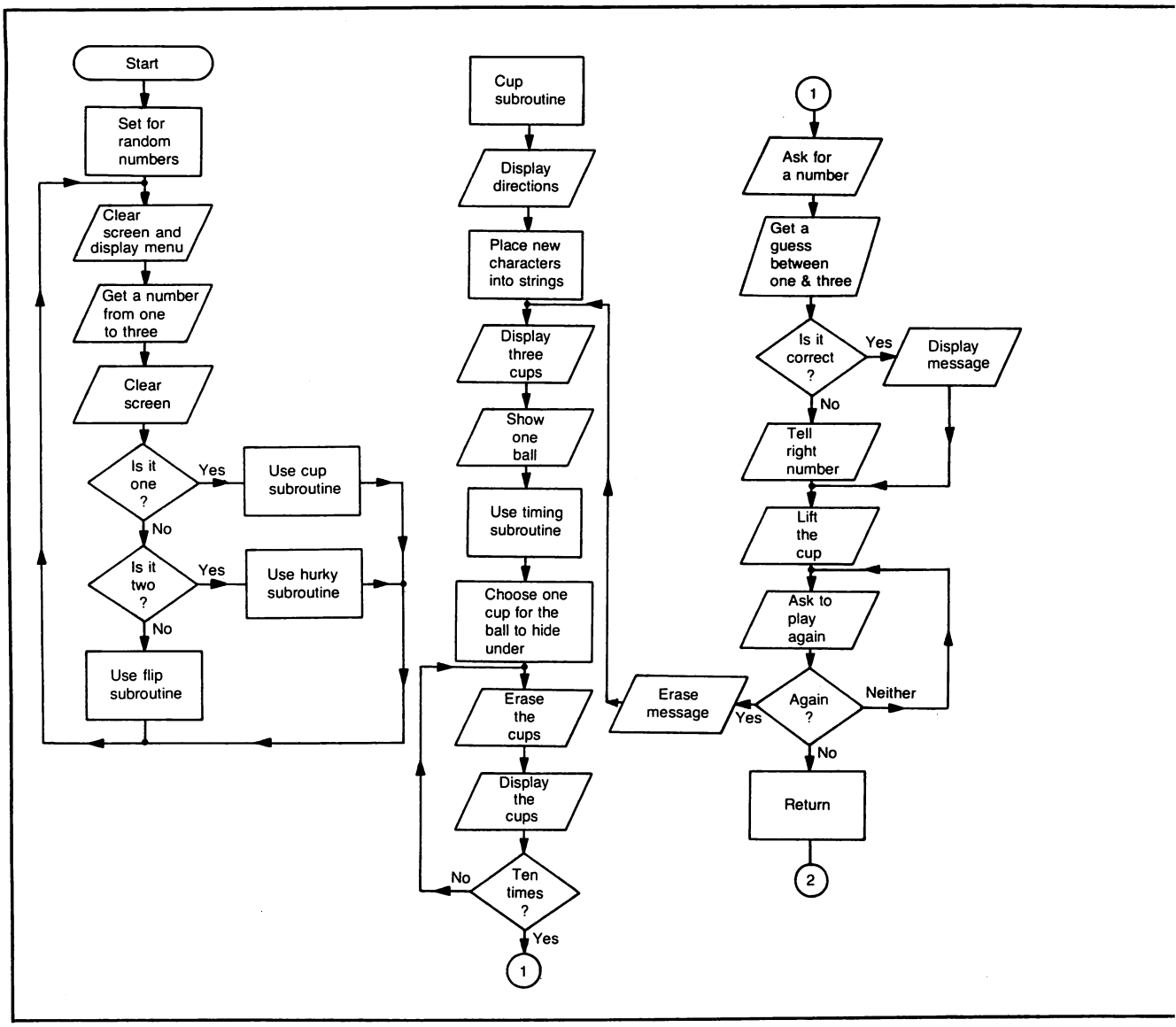
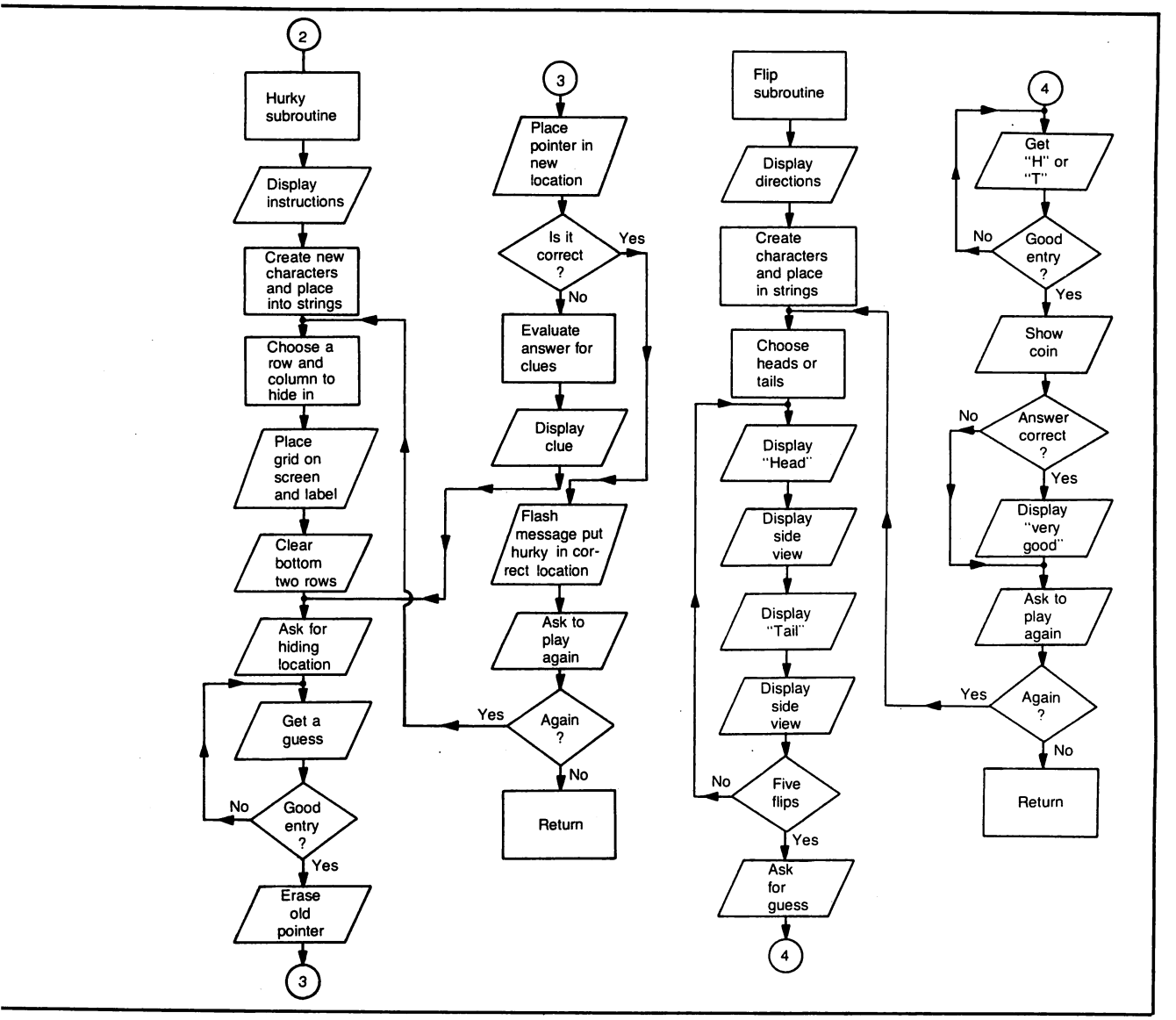


Fig. 12-2. Flowchart for Listing 12-2 Guess.

have this line in this program. Without it, the computer would continue into the music subroutine.

Lines 370-400 contain the program lines for the music subroutine. The computer will begin with the value 300 and count by 10s to 400. Each value is used in the SOUND command. The tone that you hear depends on the value of TONE. Once the sound has been made, the computer returns to the program line that it left.

A variable that is used in the main program should not be used in any subroutine unless you know for certain that you will not need the currently stored value of that variable later in the program. If you are using a variable for a counter within a



subroutine, it should be reset to zero when you enter the subroutine. If it is not reset each time, it will continue to count starting at the last value that it held.

ON . . . GOSUB/RETURN

As you did with the ON . . . GOTO command, you can selectively branch to a subroutine from the main program with an ON . . . GOSUB command, where the subroutine entered is determined by the value of a variable. The computer remembers the line that the GOSUB was on, executes the subroutine, and returns to the next program line.

Be sure that all subroutines end with a RETURN statement. If the RETURN statement is not there, the computer will continue with the lines

Listing 12-2

```
100 REM LISTING 12-2
110 REM GUESS
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 RANDOMIZE ! PICK NEW RANDOM NUMBERS
140 REM PUT UP MENU & GET SELECTION
150 DISPLAY AT(10,1)ERASE ALL:"PLEASE CH
OOSE A UNIT (1-3)" :: DISPLAY AT(12,10):
"1) CUPS" :: DISPLAY AT(14,10):"2) HURKY
"
160 DISPLAY AT(16,10):"3) FLIP" :: ACCEP
T AT(10,28)BEEP VALIDATE("123")SIZE(1):U
NIT
170 CALL CLEAR ! REMOVE THE MENU
180 REM GO PERFORM UNIT PICKED AND THEN
CONTINUE AT NEXT LINE
190 ON UNIT GOSUB 220,530,960
200 GOTO 150 ! GO BACK TO MENU
210 REM CUPS INSTRUCTIONS
220 PRINT "HERE ARE 3 CUPS.  I WILL":"PL
ACE A BALL UNDER ONE OF":"THEM, AND MIX
THEM UP.  YOU WILL TELL ME WHERE THE BAL
L IS."
230 PRINT :"READY??? GOOD...LET'S GO!!":
 : : : : : : : : : : : : : :
240 REM CREATE CHARACTERS TO FORM CUPS &
 BALL FROM UNUSED CHARACTER CODES
250 CALL CHAR(128,"FFFFC0C0C0C0C0C0"):::
CALL CHAR(129,"FFFF"):: CALL CHAR(130,"F
FFF030303030303"):: CALL CHAR(131,"3C7EF
FFFFFFF7E3C")
260 CUP$=CHR$(128)&CHR$(129)&CHR$(130)::
 BALL$=CHR$(131):: ERASE$="    " :: TIMEA
DJ=15
270 REM DISPLAY 3 CUPS IN A ROW
280 DISPLAY AT(15,8):CUP$ :: DISPLAY AT(
15,13):CUP$ :: DISPLAY AT(15,18):CUP$
290 REM ERASE 1 CUP, REDISPLAY IT 1 ROW
HIGHER, PUT BALL ON ROW BELOW RAISED CUP
, & DELAY A FEW SECONDS
300 DISPLAY AT(15,8)SIZE(3):ERASE$ :: DI
SPLAY AT(14,8):CUP$ :: DISPLAY AT(15,9)S
IZE(1):BALL$
310 TIME=200*TIMEADJ :: GOSUB 1360 ! GIV
E TIME TO READ INSTRUCTIONS
```

continued on page 86

following the subroutine until it comes to the end of the program, finds another return, or crashes. Also, if you place your subroutines at the end of the program, be sure an END statement is between the end of the program and the subroutines. If the program does not end, it will continue into the subroutine until it finds the RETURN statement, and then crash with an error message. The program flowcharted in Fig. 12-2 and listed in Listing 12-2 demonstrates the use of selective subroutines.

Listing 12-2

Line 130 uses the RANDOMIZE command so that the computer will choose different numbers every time it's asked to pick a number.

Lines 150-160 display a menu on the screen. In this program you can choose from three different units. The following sequence of events occurs before the choice is made: the computer erases the entire screen, displays the unit names and numbers, beeps, clears one space for the entry, and then waits for a number to be entered. Only the numbers 1, 2, and 3 will be accepted.

Line 170 clears the screen.

Line 190 directs the computer to the correct subroutine. The subroutine that it will go to is determined by the value of the UNIT variable.

Line 200 sends the computer back to line 150 to place the menu on the screen again. When the program returns from one of the three selected subroutines, it will return to this line. The GOTO command is necessary to keep the computer from continuing into the first subroutine of the program.

Lines 220-230 print the instructions on the screen. Be sure that all 16 colons in line 230 are entered. These colons move the instructions up to the top of the screen.

Line 250 uses the CHAR command to create new characters (depicted in Fig. 12-3). These characters will be the cup and the ball. We will cover the procedures for creating new character sets in detail in a later chapter. In the CHAR command the letters after the comma tell the computer how the characters should look. If these letters are not entered correctly, your cups may not look right.

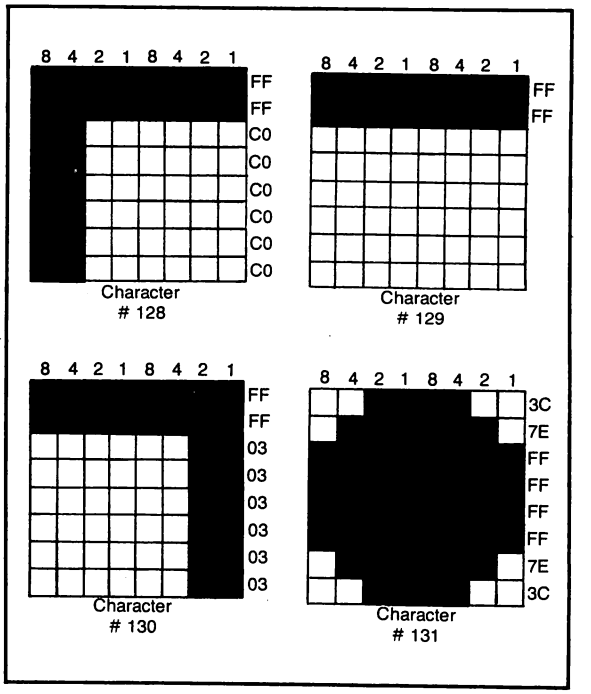


Fig. 12-3. Characters used in cups routine Listing 12-2.

Line 260 places the new characters in CUP$ and BALL$. It is much easier and smoother to print a string on the screen. The string ERASE$ is three spaces. When we want to remove a cup from the screen, we will print ERASE$ over the location of the cup. The TIMEADJ variable is set to 15. We will use this variable for timing routines.

Line 280 prints all three cups on the screen using the DISPLAY AT command. At first, all three cups will be on the same row or line.

Line 300 uses the DISPLAY AT command to erase the first cup and reprint it at a slightly higher location on the screen. BALL$ is printed under the cup. When the computer runs this line, it gives the illusion of picking up the cup and showing the ball under it.

Line 310 sets the TIME variable by multiplying 200 by the value of TIMEADJ. The computer is then directed to the subroutine that begins with line 1360. This subroutine is the timing routine and will be used by other units in this program. This subroutine is used to give the user enough time to read the instructions that are on the screen.

```
320 REM PUT CUP OVER BALL - ONLY CUP SHO
WS
330 DISPLAY AT(14,8):ERASE$ :: DISPLAY A
T(15,8):CUP$
340 HIDE=INT(RND*3)+1 ! GET NUMBER WHERE
 TO HIDE BALL
350 REM ILLUSION OF MOVING THE CUPS
360 FOR MOVEMENT=1 TO 10
370 DISPLAY AT(15,8):ERASE$
380 DISPLAY AT(15,8):CUP$ :: DISPLAY AT(
15,13):CUP$ :: DISPLAY AT(15,18):CUP$
390 NEXT MOVEMENT
400 REM ASK FOR GUESS
410 DISPLAY AT(21,1):"WHERE IS THE BALL
(1,2,3)?" :: ACCEPT AT(21,28)BEEP VALIDA
TE("123")SIZE(1):GUESS
420 REM DETERMINE IF GUESS IS CORRECT &
TELL PLAYER
430 IF GUESS=HIDE THEN DISPLAY AT(22,7):
"THAT'S RIGHT!!!" :: GOTO 460
440 DISPLAY AT(22,5):"IT WAS UNDER CUP";
HIDE
450 REM RAISE APPROPRIATE CUP & SHOW BAL
L UNDERNEATH
460 DISPLAY AT(15,8+(HIDE-1)*5)SIZE(3):E
RASE$ :: DISPLAY AT(14,8+(HIDE-1)*5):CUP
$ :: DISPLAY AT(15,9+(HIDE-1)*5)SIZE(3):
BALL$
470 REM FIND OUT IF PLAYER WANTS TO PLAY
 AGAIN - ONLY ACCEPT A 'Y' OR 'N' ANSWER
480 DISPLAY AT(24,1):"WANT TO PLAY AGAIN
 (Y/N) ?" :: ACCEPT AT(24,28)BEEP VALIDA
TE("YN")SIZE(1):ANSWER$ :: IF ANSWER$=""
 THEN 480
490 IF ANSWER$="N" THEN RETURN ! GO BACK
 TO MENU
500 REM ERASE OLD GAME BEFORE GOING BACK
 TO PLAY IT AGAIN - LEAVE INSTRUCTIONS
510 FOR ROW=14 TO 24 :: DISPLAY AT(ROW,1
):" " :: NEXT ROW :: TIMEADJ=1 :: GOTO 2
80
520 REM HURKY INSTRUCTIONS
530 DISPLAY AT(1,1):"HURKY IS VERY SHY.
 HE LIVESIN A 10 X 10 GRID.  TRY TO" ::
```

continued on page 88

Line 330 uses the ERASE$ again to remove the first cup from the screen. The cup is then reprinted in its original position, covering the ball.

Line 340 chooses a random number from one to three to select the cup that the ball will be placed under.

Lines 360-390 create the illusion that the cups are moving on the screen. The computer erases the cups from the screen, then reprints them. The speed at which the computer erases and reprints the cups gives the viewer the illusion that the cups are moving on the screen. The cups are erased and reprinted ten times.

Line 410 asks the user to guess which cup the ball is under. The computer will only accept the numbers one, two, or three. This entry will be stored in the GUESS variable.

Line 430 checks the entry against the number that the computer picked. If both numbers are the same, the computer will congratulate the user and go on to line 460.

Line 440 will tell the user where the ball was if the guess was wrong.

Line 460 erases the cup that the ball was under, reprints it in a slightly higher location on the screen, and prints the ball under the cup. This can be done with one formula. The cups were originally printed five columns apart on the screen. The first cup is in column 8, the second in 13, and the third in 18. To find out which cup should be erased, we subtract one from the number of the cup that will be raised. This number is stored in the HIDE variable. Because the cups are five columns apart, we multiply 5 times the new number. Then 8 is added to the number because the first cup is 8 columns from the left edge of the screen. If the cups were not placed the same number of rows apart, we could not use such a simple formula to determine where the three cups were on the screen.

Line 480 asks the user to play again. Only the letters "Y" and "N" will be accepted by the computer. The computer will remain at this line until a "Y" or "N" is entered.

Line 490 checks the entry for the letter "N." If the "N" was entered, the computer returns to the main menu.

Line 510 is used when a "Y" is entered. The computer does not have to check the entry for a "Y" because only the two letters, "Y" and "N," were accepted in the first place. This line erases the cups from the screen, changes the value of TIMEADJ to one and directs the computer to line 280. This game will continue until the user enters an "N," or presses the FCTN and 4 keys.

Lines 530-550 begin the hurky unit. These three lines print the directions on the screen.

Lines 570-580 change some of the characters on the character set. DOT$ contains an entire row of dots. The RPT$ command tells the computer that we want to place character number 131 (Fig. 12-4) followed by a space in DOT$ 10 times. HURKY$ contains the new character that represents HURKY. The character that represents which position was chosen is in POINTER$.

Line 600 picks a hiding place for HURKY. We need to choose both a row and a column. The column that HURKY will be hiding in is stored in the HURKCOL variable. The row is stored in the HURKROW variable. The computer will pick a

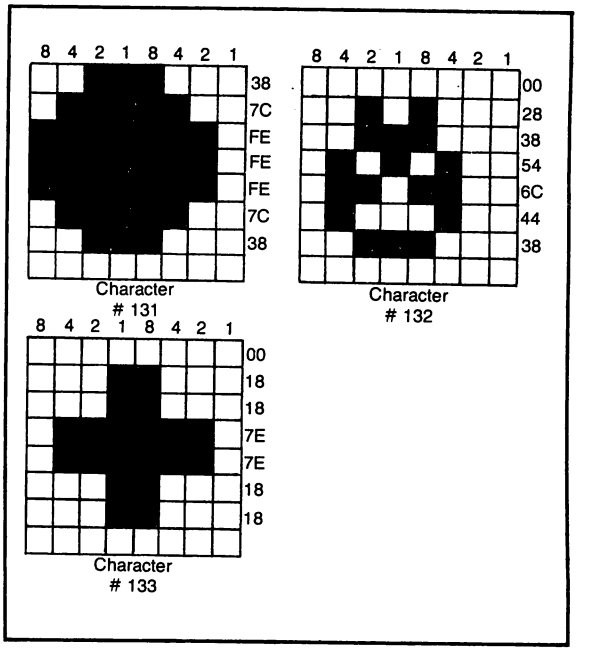


Fig. 12-4. Characters used in hurky Listing 12-2.

```
DISPLAY AT(3,1):"FIND 'HURKY' BY ENTERIN
G THECOLUMN AND ROW"
540 DISPLAY AT(4,16):"NUMBER WHERE" :: D
ISPLAY AT(5,1):"YOU THINK HE IS - LIKE T
HIS 3,4.  IF YOU DID NOT GUESS"
550 DISPLAY AT(7,1):"WHERE  'HURKY' IS,
YOU WILL BE TOLD WHICH WAY TO GO."
560 REM CREATE PLAYING CHARACTERS FROM U
NUSED CHARACTER CODES
570 CALL CHAR(131,"387CFEFEFE7C38"):: DO
T$=RPT$(CHR$(131)&" ",10):: CALL CHAR(13
2,"002838546C4438"):: HURKY$=CHR$(132)
580 CALL CHAR(133,"0018187E7E1818"):: PO
INTER$=CHR$(133)
590 REM PICK A HIDING PLACE FOR HURKY &
PLAYERS STARTING LOCATION
600 HURKCOL=INT(RND*10)+1 :: HURKROW=INT
(RND*10)+1 :: OLDCOL=1 :: OLDROW=1
610 REM PUT UP GRID WITH ROW NUMBERS
620 COLADJ=7 :: FOR ROW=11 TO 20 :: DISP
LAY AT(ROW,COLADJ-4):11-(ROW-10):: DISPL
AY AT(ROW,COLADJ):DOT$ :: NEXT ROW
630 REM PUT IN COLUMN NUMBERS
640 COUNT=1 :: FOR COL=6 TO 24 STEP 2 ::
 DISPLAY AT(ROW,COL):COUNT :: COUNT=COUN
T+1 :: NEXT COL
650 REM SHOW NORTH, SOUTH, EAST, & WEST
DIRECTIONS
660 DISPLAY AT(10,16):"N" :: DISPLAY AT(
15,2)SIZE(1):"W" :: DISPLAY AT(15,27):"E
" :: DISPLAY AT(ROW+1,16):"S"
670 REM ERASE BOTTOM 2 ROWS
680 DISPLAY AT(23,1):" " :: DISPLAY AT(2
4,1):" "
690 REM USE A STRING TO GET 2 VALUES WIT
H 1 ACCEPT STATEMENT
700 DISPLAY AT(24,5):"WHERE AM I HIDING?
" :: ACCEPT AT(24,24)BEEP VALIDATE("0123
456789,")SIZE(5):GUESS$
710 REM BREAK THE STRING DOWN INTO 2 NUM
ERIC VALUES
720 COMMA=POS(GUESS$,",",1):: COLGUESS=V
AL(SEG$(GUESS$,1,COMMA-1)):: ROWGUESS=VA
L(SEG$(GUESS$,COMMA+1,2))
```

continued on page 90

number between one and ten for both the row and the column because the grid is made up of ten rows and ten columns. The OLDCOL and OLDROW variables are both set to one.

Line 620 prints the grid on the screen. The COLADJ variable is set to seven. This is the position of the first dot of the grid. The FOR . . . NEXT loop counts from 11 to 20. These are the rows that the dots will be printed on. The first DISPLAY AT command finds the column that the row number will be printed in. Four is subtracted from the value of COLADJ. Then, ten is subtracted from the value of ROW and this difference is subtracted from 11. The resulting number is the actual row number. The reason for all this subtraction is, the FOR . . . NEXT loop is counting forward, but the top row of the grid is the 10th row, the one under it is the 9th, the next 8th, and so on. So, we have to subtract the number that it is (the first row is one) from 11 to arrive at the row number that should be printed on the screen. The next DISPLAY AT command prints the entire row of dots on the screen. The loop continues until the entire grid is on the screen.

Line 640 prints the column numbers across the bottom of the grid. The COUNT variable is the number of the column. The FOR . . . NEXT loop COL begins with the 6th column and ends with the 24th. The loop steps by two, using only the even-numbered columns because there is a space between each dot. One is added to COUNT to keep the column numbers accurate.

Line 660 places the points of the grid, N, S, E, and W, on the screen.

Line 680 erases the last two rows of the screen. These two rows are used to accept numbers from the user and display the clues for the next guess.

Line 700 asks the user to enter the column and row of the location of HURKY. The column number is entered first, then a comma, then the row column. The computer stores this entry in the GUESS$ variable.

Line 720 takes GUESS$ and removes the column number and the row number. First the computer finds the comma with the POS command. The COMMA variable will hold the position of the comma in GUESS$. The COLGUESS variable will hold the number of the column entered. This number is obtained by taking the value of GUESS$ from the first character to the character just before the comma. ROWGUESS is the row that has been entered. The computer finds this value by taking the value of the string from the position just after the comma to the end of the string. The maximum number of characters is two.

Line 740 tests the ROWGUESS and COLGUESS variables to make sure that their values are between one and ten. If either variable is less than one or greater than ten, the computer will be sent to line 700 to wait for another set of coordinates.

Line 760 places the pointer at the column and row position that was entered in line 700. First the old pointer (if there is one) is erased and replaced with a dot. The first time an entry is made, there is no dot to replace. On every move after the first, the pointer will be replaced with a dot. Then the pointer is printed at the new location.

Line 780 compares the column guessed with the column that HURKY is in and the row that was guessed with the row that HURKY is in. If, and only if, both the row and column match the computer will be directed to line 890. HURKY will be found when both the row and column guessed match the row and column that HURKY is in.

Line 800 begins the lines that give the clues to help find HURKY. In this line the word "GO" is printed on the screen. The string variable DIRECTION$ is cleared. This is where the clue will be stored.

Line 810 checks the row that HURKY is in against the row that was guessed. If HURKY's row is less than the guessed row, then the player will have to guess a smaller number so the word "SOUTH' is stored in DIRECTION$.

Line 820 compares the rows again. This time, if the row that HURKY is in is greater than the row guessed, the player will have to try a larger number, so "NORTH" is stored in the string variable DIRECTION$. If the row guessed is the same row that HURKY is in, neither "NORTH" nor "SOUTH" will be stored in DIRECTION$.

Line 820 compares the rows again. This time, if the

```
730 REM IF ENTERED VALUES ARE ILLEGAL GO
 BACK & GET NEW VALUES
740 IF ROWGUESS<1 OR COLGUESS<1 OR ROWGU
ESS>10 OR COLGUESS>10 THEN 700
750 REM PLACE DOT AT OLD LOCATION AND CR
OSS AT NEW LOCATION
760 DISPLAY AT(21-OLDROW,OLDCOL*2+5)SIZE
(1):CHR$(131):: DISPLAY AT(13-(ROWGUESS-
8),COLGUESS*2+5)SIZE(1):POINTER$
770 REM IF PLAYER'S GUESS IS CORRECT GO
TO MESSAGE
780 IF COLGUESS=HURKCOL AND ROWGUESS=HUR
KROW THEN 890
790 REM PLAYER'S GUESS IS INCORRECT GIVE
 APPROPRIATE DIRECTIONS
800 DISPLAY AT(23,12):"GO" :: DIRECTION$
=""
810 IF HURKROW<ROWGUESS THEN DIRECTION$=
"SOUTH"
820 IF HURKROW>ROWGUESS THEN DIRECTION$=
"NORTH"
830 IF HURKCOL<COLGUESS THEN DIRECTION$=
DIRECTION$&"WEST"
840 IF HURKCOL>COLGUESS THEN DIRECTION$=
DIRECTION$&"EAST"
850 DISPLAY AT(23,15):DIRECTION$ :: TIME
=2000 :: GOSUB 1360
860 REM REPLACE OLD PLAYER'S LOCATION WI
TH CURRENT GUESS & GO BACK & TRY AGAIN
870 OLDCOL=COLGUESS :: OLDROW=ROWGUESS :
: GOTO 700
880 REM FLASH THE MESSAGE AND HURKY
890 TIME=200 :: FOR COUNT=1 TO 5 :: DISP
LAY AT(23,7):"YOU FOUND ME!!!" :: DISPLA
Y AT(21-HURKROW,HURKCOL*2+5)SIZE(1):HURK
Y$ :: GOSUB 1360
900 DISPLAY AT(23,5):" " :: DISPLAY AT(2
1-HURKROW,HURKCOL*2+5)SIZE(1):ERASE$ ::
GOSUB 1360 :: NEXT COUNT
910 REM LEAVE HURKY ON SCREEN
920 DISPLAY AT(21-HURKROW,HURKCOL*2+5)SI
ZE(1):HURKY$
930 REM FIND OUT IF PLAYER WANTS TO PLAY
 AGAIN & IF SO, REPEAT GAME, OTHERWISE,
```

continued on page 92

row that HURKY is in is greater than the row guessed, the player will have to try a larger number, so "NORTH" is stored in the string variable DIRECTION$. If the row guessed is the same row that HURKY is in, neither "NORTH" nor "SOUTH" will be stored in DIRECTION$.

Line 830 compares the column guessed with the column that HURKY is in. If the column that HURKY is in is less than the column that was guessed, then the player will have to choose a smaller number. The word "WEST" is added to the direction that is in DIRECTION$. This way, the player can be directed to go Northwest or Southwest.

Line 840 compares the columns to see if the player should move to the East. If this is the case, the direction "EAST" will be added to the direction in DIRECTION$. If the column guessed is the same as the column that HURKY is in, neither "WEST" nor "EAST" will be added to the DIRECTION$.

Line 850 prints the clue held in DIRECTION$ on the screen. The computer will use the same timing loop in line 1360 that it used in the last unit.

Line 870 places the number of the column that the pointer is in in the OLDCOL variable and the row that the pointer is in in the OLDROW variable. Now it will know where to place the dot after the next guess is made. The computer is sent to line 700 to wait for another guess.

Lines 890-900 begin the end of the program. The computer is directed to this line if the player guesses both the row and the column that HURKY is in. The TIME variable is set to 200 for a fast timing loop. The FOR . . . NEXT loop counts from one to five. Near the bottom of the screen "YOU FOUND ME!!!" is displayed. At the location where HURKY has been hiding, "HURKY" is printed. The message and "HURKY" is printed and erased five times.

Line 920 prints "HURKY" on the screen at the correct location.

Line 940 asks to play again. Only the letters "Y" and "N" can be accepted by the computer. This time, after the computer checks ANSWER$ for a "Y," it will return if an "N" or nothing is entered. If the "Y" is entered, the computer will go to line 600 for another game.

Lines 960-970 begin the third module—FLIP. The instructions are printed on the screen.

Lines 990-1020 create the characters that will be used in this unit. There are 11 new characters (Fig. 12-5). CH$ will contain the characters that display the edge or side view of the coin.

Line 1030 places the characters that make up the top part of the coin in CT$, the bottom part of the coin in CB$, the middle of the coin on the heads side in MH$, and the middle of the coin on the tails side in MT$. By printing these strings on the screen in the proper order, we will be able to simulate a coin flipping on the screen.

Line 1040 chooses a random number. The computer will only pick a one or a two since there are only two sides to a coin.

Line 1060 sets the TIME variable to five. This variable is used in the timing loop. The FOR . . . NEXT loop begins at this line. The coin will flip five times.

Lines 1070-1130 print the coin in all of its positions. First the coin will be shown with the "H" for head on it, then the middle section will be removed and only the top third and bottom third will be on the screen. At line 1090, only the side view on the coin will be on the screen. The top third and bottom third will be printed again, then the back or tail side of the coin. This same procedure will occur one more time until the head is displayed again and the coin has flipped completely.

Line 1140 continues this loop until the coin has flipped five times.

Line 1150 prints only the top third and bottom third of the coin. The computer is then sent to the timing subroutine.

Line 1170 prints a blank coin on the screen. Now you have to guess what it is.

Line 1190 asks what is it—heads or tails. The VALIDATE option will accept only an "H" or a "T" as your answer. Your guess is stored in GUESS$. The string is checked to make sure that a letter was entered. If the string is empty or null, this line will be repeated.

Line 1210 directs the computer to the correct line to display the face of the coin. If the "N" variable

```
GO BACK TO MENU
940 DISPLAY AT(24,1):"WANT TO PLAY AGAIN
 (Y/N)?" :: ACCEPT AT(24,27)BEEP VALIDAT
E("YN")SIZE(1):ANSWER$ :: IF ANS ER$="Y"
 THEN 600 ELSE RETURN
950 REM FLIP INSTRUCTIONS
960 DISPLAY AT(1,1):"I WILL FLIP A COIN.
  YOU" :: DISPLAY AT(2,1):"MUST GUESS WH
AT IT WILL BE." :: DISPLAY AT(3,1):"ENTE
R AN 'H' FOR HEADS -"
970 DISPLAY AT(4,1):"A 'T' FOR TAILS."
980 REM CREATE CHARACTERS TO MAKE UP THE
 COIN IN 3 POSITIONS FROM UNUSED CHARACT
ER CODES
990 CALL CHAR(133,"0000030C10102020"):: 
CALL CHAR(134,"828282FE82828282"):: CALL
 CHAR(135,"0000C03008080404")
1000 CALL CHAR(136,"4040808080804040"):: 
 CALL CHAR(137,"0202010101010202"):: CAL
L CHAR(138,"202010100C03")
1010 CALL CHAR(139,"0404080830C0"):: CAL
L CHAR(140,"FE101010101010"):: CALL CHAR
(141,"000000000000C33C"):: CALL CHAR(142
,"3CC3")
1020 CALL CHAR(143,"0000FFA5A5FF"):: CH$
=CHR$(143)&CHR$(143)&CHR$(143)
1030 CT$=CHR$(133)&CHR$(142)&CHR$(135):: 
 CB$=CHR$(138)&CHR$(141)&CHR$(139):: MH$
=CHR$(136)&CHR$(134)&CHR$(137):: MT$=CHR
$(136)&CHR$(140)&CHR$(137)
1040 N=INT(RND*2)+1 ! PICK A RANDOM NUMB
ER TO REPRESENT 'HEADS' OR 'TAILS'
1050 REM DISPLAY ILLUSION OF A FLIPPING
COIN
1060 TIME=5 :: FOR POSITION=1 TO 5
1070 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):MH$ :: DISPLAY AT(12,10):CB$ :: 
GOSUB 1360
1080 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):CB$ :: DISPLAY AT(12,10):" " :: 
GOSUB 1360
1090 DISPLAY AT(10,10):" " :: DISPLAY AT
(11,10):CH$ :: GOSUB 1360
1100 DISPLAY AT(10,10):CT$ :: DISPLAY AT
```

continued on page 94

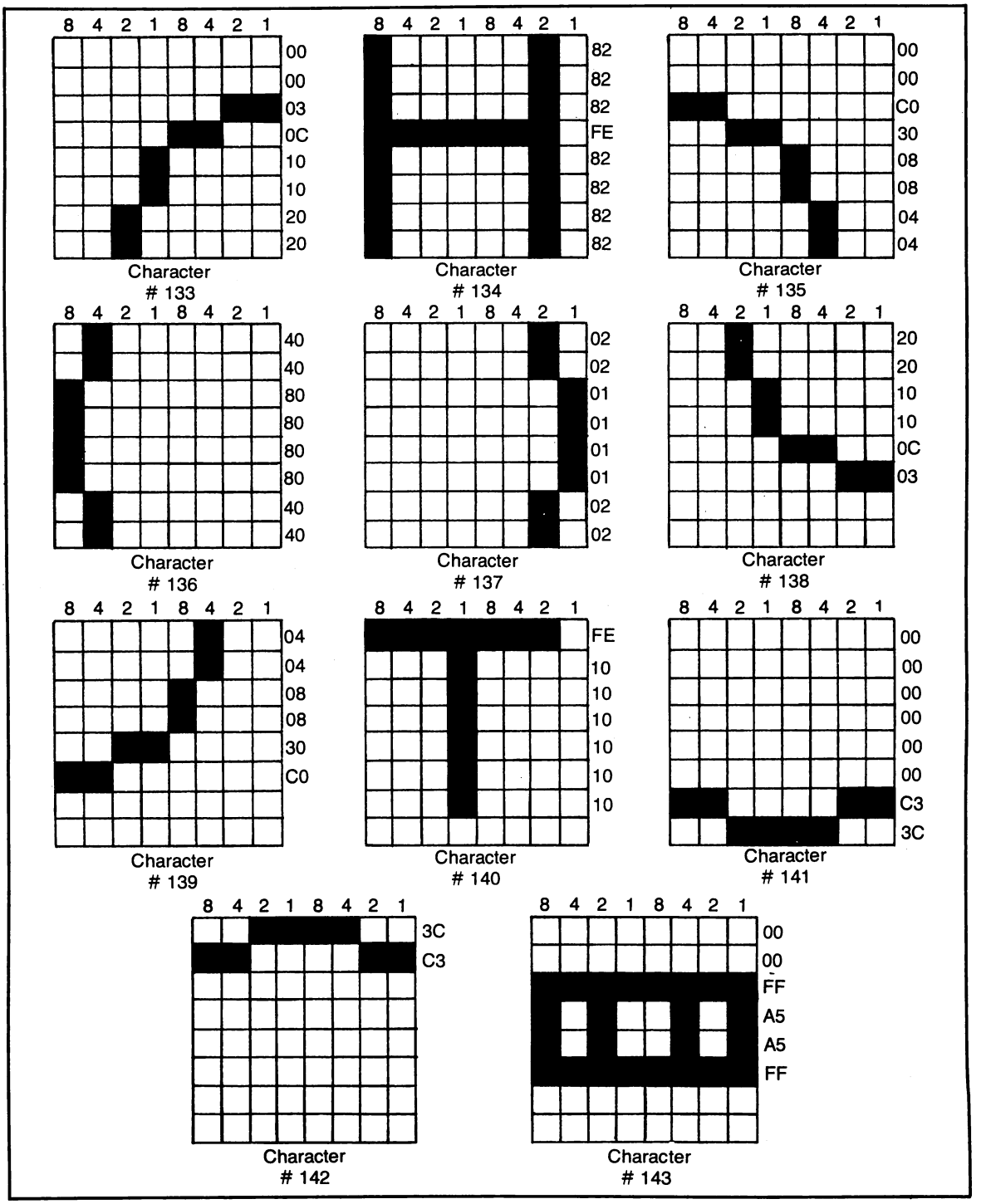


Fig. 12-5. Characters used in flip Listing 12-2.

```
(11,10):CB$ :: DISPLAY AT(12,10):" " ::
GOSUB 1360
1110 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):MT$ :: DISPLAY AT(12,10):CB$ ::
GOSUB 1360
1120 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):CB$ :: DISPLAY AT(12,10):" " ::
GOSUB 1360
1130 DISPLAY AT(10,10):" " :: DISPLAY AT
(11,10):CH$ :: GOSUB 1360
1140 NEXT POSITION
1150 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):CB$ :: GOSUB 1360
1160 REM DISPLAY A BLANK COIN ON SCREEN
1170 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):CHR$(136)&" "&CHR$(137):: DISPLA
Y AT(12,10):CB$
1180 REM GET PLAYER'S GUESS - ONLY ACCEP
T A 'H' OR 'T'
1190 DISPLAY AT(15,5):"WHAT IS IT (H-T)?
" :: ACCEPT AT(15,23)BEEP VALIDATE("HT")
SIZE(1):GUESS$ :: IF GUESS$="" THEN 1190
1200 REM GO TO CORRESPONDING LINE BASED
ON NUMBER THAT DETERMINES 'HEAD' OR 'TAI
L'
1210 ON N GOTO 1230,1240
1220 REM DISPLAY APPROPRIATE COIN FACE
1230 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):MH$ :: DISPLAY AT(12,10):CB$ ::
GOTO 1260
1240 DISPLAY AT(10,10):CT$ :: DISPLAY AT
(11,10):MT$ :: DISPLAY AT(12,10):CB$
1250 REM DETERMINE IF GUESS WAS CORRECT
& IF SO, GIVE MESSAGE
1260 IF N=1 AND GUESS$="H" THEN 1300
1270 IF N=2 AND GUESS$="T" THEN 1300
1280 REM WRONG GUESS - GO SEE IF PLAYER
WANTS TO TRY AGAIN
1290 GOTO 1310
1300 DISPLAY AT(20,8):"VERY GOOD!!!"
1310 DISPLAY AT(23,1):"WANT TO PLAY AGAI
N (Y/N)?" :: ACCEPT AT(23,27)BEEP VALIDA
TE("YN")SIZE(1):ANSWER$
1320 REM IF PLAYER WANTS TO PLAY AGAIN,
```

```
ERASE OLD MESSAGE LINES FIRST & THEN REP
EAT
1330 IF ANSWER$="Y" THEN DISPLAY AT(15,1
):" " :: DISPLAY AT(20,8):" " :: DISPLAY
 AT(23,1):" " :: GOTO 1040
1340 RETURN ! PLAYER WANTS TO GO BACK TO
 MENU
1350 REM TIMING SUBROUTINE
1360 FOR DELAY=1 TO TIME :: NEXT DELAY :
: RETURN
```

is equal to one, the computer will be directed to line 1230; otherwise it will be directed to line 1240.

Line 1230 prints the "H" on the coin—if "N" is a one then heads was chosen. The program continues with line 1260.

Line 1240 prints the "T" for tails on the screen.

Line 1260 checks to see if the computer picked heads ("N" is equal to 1) and the player picked heads. If the player did, the computer is directed to line 1300.

Line 1270 checks to see if both the computer and the player chose tails. If both did, the computer will go on to line 1300.

Line 1290 sends the computer to line 1310, skipping line 1300 so that the player will not be congratulated.

Line 1300 congratulates the player for making the correct guess.

Line 1310 asks to play again. Only the letters "Y" and "N" will be accepted and stored in ANSWER$.

Line 1330 checks to see if ANSWER$ contains a "Y." If it does, the messages will be erased from the screen and the computer will be directed to line 1040 for another flip.

Line 1340 sends the computer back to the main menu. It will return to the main menu if the N key was pressed, or if the ENTER key was pressed without entering any other letter.

Line 1360 is the timing subroutine that had been used by all three units in this program.

CALLING A SUBROUTINE

CALL

Throughout the last program (Listing 12-2) and in several other programs in this book, the CALL command has been used. It is never used alone. Another word or command follows it. The CALL command uses subroutines that are built into the TI-99/4A. The CALL CLEAR command directs the computer to the built-in subroutine that clears the screen and then returns to the main program. Other subroutines used were CALL KEY to find out what key has been pressed and CALL CHAR to redefine a character. There are many more subroutines like these built into the TI-99/4A, and we will examine them closely in other chapters.

The CALL command is not restricted to the built-in subroutines. You can write your own subroutines that can be called by name, just like the subroutines that TI developed. In the next program we will develop two subroutines that will be called from the main program.

You may be wondering why you would want to use the CALL command when a GOSUB seems to do the same thing. There are a few differences between a subroutine that is used with the GOSUB command. The variables are used in the main program. With the CALL command, the same variable can be used in the main program and in the subroutine. The computer will remember what that variable's value should be in the main program and

keep it separate from the variable's value in the subroutine.

The subroutine that you develop to use with the CALL command can be saved to disk under its own name, then *merged with* or joined into the main program when needed.

Values can also be passed and used in a subroutine that is called with the CALL command.

DEVELOPING A SUBROUTINE

SUB

In order to use the CALL command, subroutines must be developed. How will the computer know where this subroutine is when you ask for it? Listing 12-3 is a subroutine that will be used in a future program. If you are using a cassette recorder

Listing 12-3

```
980 REM LISTING 12-3
990 REM BY A.R.SCHREIBER FOR TAB BOOKS
1000 REM SIMULATED SCROLL
1010 REM THE "CALL" COMMAND SHOULD PASS
3 VARIABLES TO THIS SUB PROGRAM AS FOLLO
WS:
1020 REM "STARTROW" IS THE TOP ROW WHERE
 INFORMATION CAN BE PRINTED AND IS ASSIG
NED TO "ROWREF" IN THIS SUB PROGRAM.  "S
TARTROW'S" VALUE IS
1030 REM NOT ALTERED.  "COUNT" IS USED I
N BOTH PROGRAMS AS THE ITEM NUMBER AND A
RRAY ELEMENT NUMBER.  IT'S VALUE IS NOT
CHANGED BY THE SUB
1040 REM PROGRAM.  "LIST$()" IS THE ARRA
Y OF ALL THE ITEMS TYPED.  IT TOO, IS NO
T ALTERED BY THIS SUB PROGRAM.
1050 REM "NEWROW" IS THE ONLY COMMON VAR
IABLE THAT THE SUB PROGRAM DOES CHANGE.
HOWEVER, A VARIABLE BY THE SAME NAME IN
THE MAIN PROGRAM
1060 REM WILL BE UNAFFECTED BY THIS CHAN
GE IN THE SUB PROGRAM.  VARIABLES USED I
N A SUB PROGRAM ARE TREATED SEPARATELY U
NLESS NAMED IN THE
1070 REM CALL OPTION LIST.
1080 SUB SCROLL(ROWREF,COUNT,LIST$())
1090 FOR NEWROW=ROWREF TO 23
1100 DISPLAY AT(NEWROW,COL):STR$(COUNT-2
3+NEWROW);".."
1110 DISPLAY AT(NEWROW,COL+6):LIST$(COUN
T-17+(NEWROW-6))
1120 NEXT NEWROW
1130 SUBEND
```

for program storage, you may want to wait until the entire program is listed before entering this subroutine. If you are using the disk, you can enter this program and save it to the disk with the MERGE option.

Listing 12-3

Lines 1000-1070 explain how this subroutine is used. There are three values that will be passed to this subroutine. This subroutine needs to know where the top row of the screen is. This is not the actual top screen row, but the place on the screen where the scroll will occur. All the information printed on the screen above this row will remain on the screen no matter how many times it is scrolled. It also needs to know the last number of the item on the screen and be given access to the LIST$ array.

Line 1080 names this subroutine. The format for a subroutine that uses the CALL command is very simple.

SUB {name} (any variables)

The line that contains the SUB command must be immediately followed by the name of the subroutine. The computer cannot find the subroutine if it is not named in the program. After the name of the subroutine are the variables that will have values passed to them from the main program. These variables are enclosed in parentheses. This subroutine is called SCROLL. The variables ROWREF, COUNT, and LIST$( ) will be used in this subroutine. The values that they contain have been passed to them through the CALL command.

Lines 1090-1120 contain the FOR . . . NEXT loop that moves the words one row up on the screen. The loop begins with the row value of NEWROW and continues to row 23. The line number is printed on the screen, then the word from LIST$. The loop continues until all the words have been moved one row up on the screen.

Line 1230 contains a SUBEND. There are only two ways to leave a SUB routine. The SUBEND is similar to the RETURN. The computer goes back to the main program. The other way is with a SUBEXIT. There can only be one SUBEND in any SUB routine.

If you have a disk drive, you can save this subroutine by typing the following line:

SAVE DSK1.SCROLL,MERGE

You can add this subroutine to any program that you are writing where you want only a portion of the screen to scroll. If you are using a cassette, you can save this program with the SAVE command. There is no MERGE option for the cassette.

WARNING: Do *not* type NEW while this routine is in memory. Just type the next routine.

The next subroutine alphabetizes a list of words. The words are stored in LIST$. The main

Listing 12-4

```
1980 REM LISTING 12-4
1990 REM BY A.R.SCHREIBER
2000 REM ALPHABETIZE SUB PROGRAM
2010 SUB ALPHABETIZE(CNT,LIST$())
2020 FOR COUNT1=1 TO CNT-1 :: FOR COUNT2
=COUNT1+1 TO CNT
2030 IF LIST$(COUNT2)<LIST$(COUNT1)THEN
TEMP$=LIST$(COUNT2):: LIST$(COUNT2)=LIST
$(COUNT1):: LIST$(COUNT1)=TEMP$
2040 NEXT COUNT2 :: NEXT COUNT1
2050 SUBEND
```

program will set aside enough memory to store up to 200 words in this string array.

Listing 12-4

Line 2010 names this subroutine. It tells the computer that this subroutine is called ALPHABETIZE. Two values are passed into this subroutine. The CNT variable contains the number of words that will be alphabetized. LIST$ contains the words that this subroutine will alphabetize.

Line 2020 begins two FOR . . . NEXT loops. The first loop counts from the first word of the string array to the next to the last word of the array. The second loop begins with the second word of the array and ends with the last word of the string array.

Line 2030 begins by comparing the second word of the string array with the first word. If the value of the second word is less than the first; that is, it comes before the first word in the alphabet, the computer will place the second word in a temporary string called TEMP$, place the first word into the second word's location in the string array, and then place the second word into the first word's location. Now the word that should come first does.

Line 2040 continues the loop. The word that is now in the first location is compared to the word in the other words, the third, fourth, fifth, and so on, until that word has been compared to all the words in the string. The loop then advances to the second word in the array and does the same thing for it and for all the others until all the words in the array have been compared to all the words that are after them in the array.

Line 2050 is the SUBEND command. After all the words have been compared, the list is in alphabetical order and the computer returns to the main program.

SAVE DSK.1 ALPHA,MERGE

If you are using the cassette recorder for the program storage, save your program using the SAVE command. Since there is no merge with cassette, you should be saving both subroutines at this time.

The next program uses both subroutines that we just typed in. If you are using the disk, you do not have to worry about having the two preceding routines in memory. If you are using the cassette, be sure that both of the previous routines are in the computer. If they are not, you can load them in from your cassette. They should have been saved together as one program.

Listing 12-5

Line 130 sets aside enough memory to hold 200 words. These are the words that the computer will alphabetize.

Line 140 clears the screen.

Lines 150-160 prints the instructions on the screen. Keep entering the words that you want the computer to alphabetize. If you do not have 200 words, press ENTER without typing in a word and the computer will know that you have finished.

Line 170 sets the STARTROW and ROW variables to six. Since both variables contain the same value, you can use one program statement with the comma between them to set both variables to the same value. Row six is the row where the first word will be printed. COL is set to 1 and the FOR . . . NEXT loop begins. This loop counts from 1 to 200, which is the number of words this program can hold.

Line 180 prints the number of the word on the screen. We are using a new command in this line—STR$. By taking the STR$ of the variable, we eliminate the spaces that are normally printed before and after the number. The period will be printed in the space immediately following the number rather than one space over. The computer waits until a word is entered. The VALIDATE option will only allow uppercase alphabetical characters to be entered. Once entered, the word will be stored in ITEM$, a string variable for temporary storage. The computer checks the entry. If ITEM$ is an empty or null string, the computer will be directed to line 220 because there are no more words to be entered. Any other word will be stored in LIST$ at the location COUNT.

Listing 12-5

```
100 REM LISTING 12-5
110 REM ALPHABETIZE
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 DIM LIST$(200)
140 CALL CLEAR
150 DISPLAY AT(1,1):"THIS PROGRAM ACCEPT
S A LIST" :: DISPLAY AT(2,1):"OF UP TO 2
00 WORDS AND" :: DISPLAY AT(3,1):"ALPHAB
ETIZES & PRINTS THEM."
160 DISPLAY AT(4,1):"TYPE AN [ENTER] ONL
Y TO END." :: DISPLAY AT(5,1):" "
170 STARTROW,ROW=6 :: COL=1 :: FOR COUNT
=1 TO 200
180 DISPLAY AT(ROW,COL):STR$(COUNT);"."
:: ACCEPT AT(ROW,COL+5)VALIDATE(UALPHA)S
IZE(22):ITEM$ :: IF ITEM$="" THEN 220 EL
SE LIST$(COUNT)=ITEM$
190 ROW=ROW+1 :: IF ROW<=24 THEN 210
200 CALL SCROLL(STARTROW,COUNT,LIST$()):
: ROW=24 ! SCROLL PREVIOUS LINES WITHOUT
 DISTURBING INSTRUCTIONS
210 NEXT COUNT
220 CALL ALPHABETIZE(COUNT-1,LIST$())
230 DISPLAY AT(1,1)ERASE ALL:"PRESS [ENT
ER] FOR NEXT SET  OF WORDS."
240 STARTROW,ROW=4 :: FOR COUNTER=1 TO C
OUNT-1 :: DISPLAY AT(ROW,COL):STR$(COUNT
ER);"." :: DISPLAY AT(ROW,COL+5):LIST$(C
OUNTER)
250 ROW=ROW+1 :: IF ROW=25 THEN ACCEPT A
T(2,12)BEEP:WAIT$ :: GOSUB 300
260 NEXT COUNTER
270 DISPLAY AT(1,1):" " :: DISPLAY AT(2,
1)BEEP:"    THAT'S ALL FOLKS!!!" :: FOR
DELAY=1 TO 5000 :: NEXT DELAY
280 END
290 REM CLEAR PART OF SCREEN
300 FOR ROW=STARTROW TO 24 :: DISPLAY AT
(ROW,1):" " :: NEXT ROW :: ROW=STARTROW
:: RETURN
310 REM SIMULATED SCROLL
320 REM THE "CALL" COMMAND SHOULD PASS 3
 VARIABLES TO THIS SUB PROGRAM AS FOLLOW
S:
```

continued on page 101

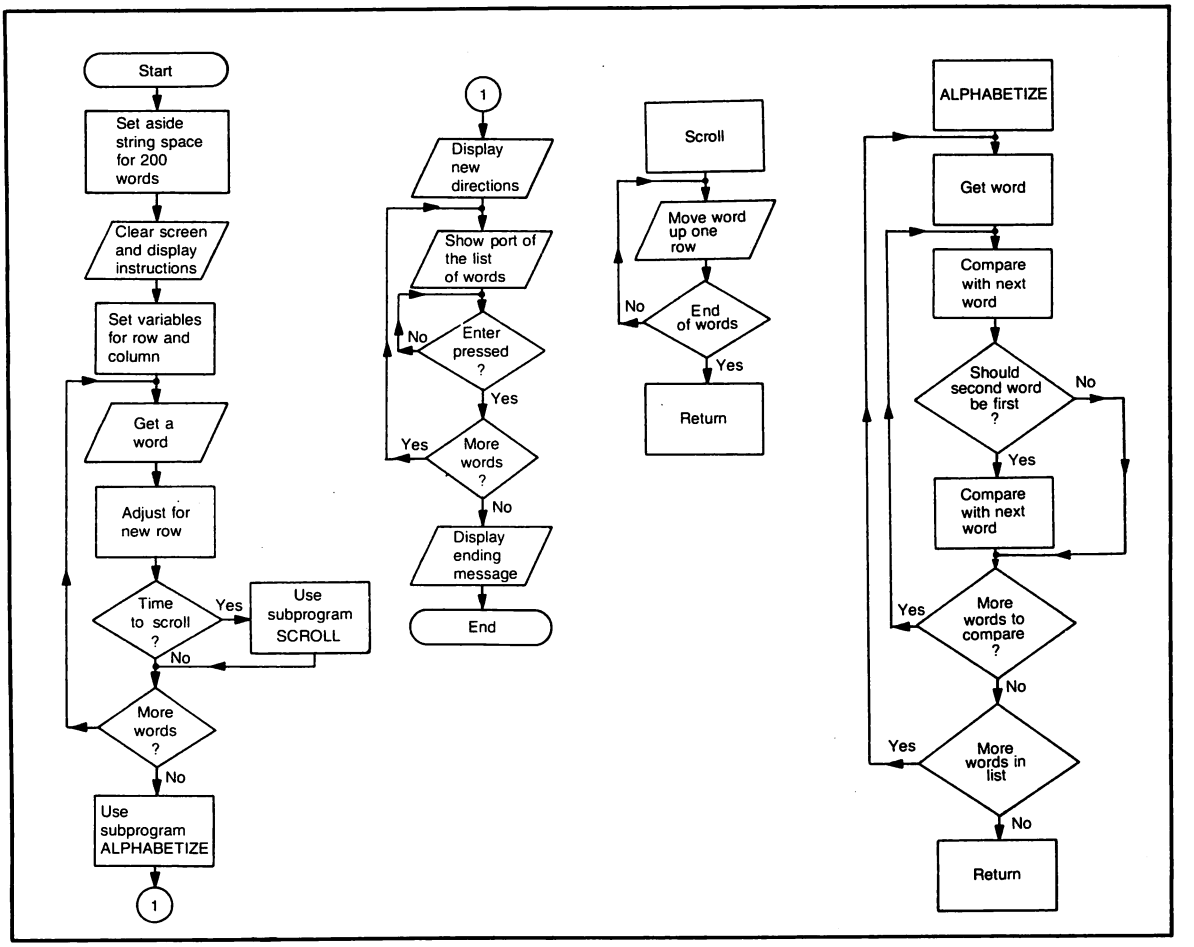


Fig. 12-6. Flowchart for Listing 12-5 Alphabetize and its subprograms. Listing 12-3 and Listing 12-4.

Line 190 adds one to the value of ROW. The next word will be entered in the next line on the screen. The value of ROW is checked for 24. If ROW is less than or equal to 24 then the computer will continue at line 210. Row 24 is the last or bottom row on the screen.

Line 200 calls the SCROLL subroutine. This was the first subroutine that we typed in. Three values will be passed to the subroutine. STARTROW is the top row that will be scrolled off the screen; COUNT is the number of the word that has just been entered, and LIST$ contains the words that have been entered. The values of these variables will be used in the subroutine. When the computer returns to this line, ROW will be reset to 24, the last line on the screen.

Line 210 continues this loop until 200 words have been entered or ENTER has been pressed without typing a word.

Line 220 calls the second subroutine, ALPHABETIZE. The two values that are passed to the subroutine are the value of COUNT minus one, and the value of LIST$. We subtract one from the value of COUNT, because COUNT will be one more than the number of words entered. If we did not enter 200 words, COUNT would be the number that the word would have been if it were entered. If we did enter 200 words, the FOR . . . NEXT loop would stop when the value of COUNT exceeded 200.

```
330 REM "STARTROW" IS THE TOP ROW WHERE
INFORMATION CAN BE PRINTED AND IS ASSIGN
ED TO "ROWREF" IN THIS SUB PROGRAM.  "ST
ARTROW'S" VALUE IS
340 REM NOT ALTERED.  "COUNT" IS USED IN
 BOTH PROGRAMS AS THE ITEM NUMBER AND AR
RAY ELEMENT NUMBER.  IT'S VALUE IS NOT C
HANGED BY THE SUB
350 REM PROGRAM.  "LIST$()" IS THE ARRAY
 OF ALL THE ITEMS TYPED.  IT TOO, IS NOT
 ALTERED BY THIS SUB PROGRAM.
360 REM "NEWROW" IS THE ONLY COMMON VARI
ABLE THAT THE SUB PROGRAM DOES CHANGE.
HOWEVER, A VARIABLE BY THE SAME NAME IN
THE MAIN PROGRAM
370 REM WILL BE UNAFFECTED BY THIS CHANG
E IN THE SUB PROGRAM.  VARIABLES USED IN
 A SUB PROGRAM ARE TREATED SEPARATELY UN
LESS NAMED IN THE
380 REM CALL OPTION LIST.
390 SUB SCROLL(ROWREF,COUNT,LIST$())
400 FOR NEWROW=ROWREF TO 23
410 DISPLAY AT(NEWROW,COL):STR$(COUNT-23
+NEWROW);"."
420 DISPLAY AT(NEWROW,COL+6):LIST$(COUNT
-17+(NEWROW-6))
430 NEXT NEWROW
440 SUBEND
450 REM ALPHABETIZE SUB PROGRAM
460 SUB ALPHABETIZE(CNT,LIST$())
470 FOR COUNT1=1 TO CNT-1 :: FOR COUNT2=
COUNT1+1 TO CNT
480 IF LIST$(COUNT2)<LIST$(COUNT1)THEN T
EMP$=LIST$(COUNT2):: LIST$(COUNT2)=LIST$
(COUNT1):: LIST$(COUNT1)=TEMP$
490 NEXT COUNT2 :: NEXT COUNT1
500 SUBEND
```

Line 230 clears the screen and instructs you to press ENTER for the next set of words. Only 21 words can be displayed on the screen at one time. The words are not erased until ENTER has been pressed.

Lines 240-260 print the words in alphabetical order on the screen. The words will begin at line 4 and continue until line 24. When the value of ROW is equal to 25, the computer will wait until ENTER has been pressed. The bottom portion of the

screen will clear and the loop will continue until all the word letters have been displayed on the screen.

Line 270 prints the ending message on the screen.

Line 280 contains the END command. This program line must be here to separate the main part of the program from the subroutine that follows it.

Line 300 is the subroutine that clears the bottom portion of the screen when the alphabetized words are being displayed. Only rows 4 through 24 will be cleared. The instructions will remain on the screen. The ROW variable is set to the value of STARTROW so that the computer will begin printing the next set of words on the 4th row. The subroutine returns to the line that sent it.

The remaining lines of the program are the two subroutines that you entered earlier. If you are using the disk, and these lines are not in the program right now, you do not have to retype them. Use the MERGE command to load the program from the disk.

```
MERGE DSK1.SCROLL
MERGE DSK1.ALPHA
```

The subroutines will be added to your program. If you are using the cassette, these routines should have been loaded before you typed in the rest of the program. The entire program can be saved to either the cassette or the disk.

Listing 12-5 shows the entire program after the subroutines have been merged. It is flowcharted in Fig. 12-6.

Chapter 13

Arithmetic Functions

TI Extended BASIC can perform any standard arithmetic function including; addition, subtraction, multiplication, division, and exponentiation (raising to a power) to name a few. When the computer solves an equation, it carries out the operations in a specific order. The *order of precedence* is listed from first performed to last below:

1. parentheses ( )
2. exponentiation (raising to a power)
3. multiplication (*) and/or division (/)
4. addition (+) and/or subtraction (-)

If you want a subtraction operation completed before multiplication, you must place the numbers and/or variables in parentheses. Below are some examples of the way the computer would solve various types of equations.

$$4+2*3-8=2$$

$$8*(53-8)+9=369$$

$$47-2^2+(4*5)=63$$

In any equation, variables can be substituted for the numbers. If a value has been assigned to the variable, the computer will use that value. If no value has been assigned, the computer will use a zero. In this chapter we will discuss the five most frequently used special functions.

SPECIAL FUNCTIONS

INT

When you want a whole number without a fraction (any numbers after the decimal point), you will use the INT(integer) command. This command ignores any numbers following the decimal point and the variable becomes a whole number. X=INT(10/3). The X variable would be equal to three rather than 3.333 . . . The program flowcharted in Fig. 13-1 and listed in Listing 13-1 shows how the INT command can be used.

Listing 13-1

Line 130 clears the screen. The amount of money

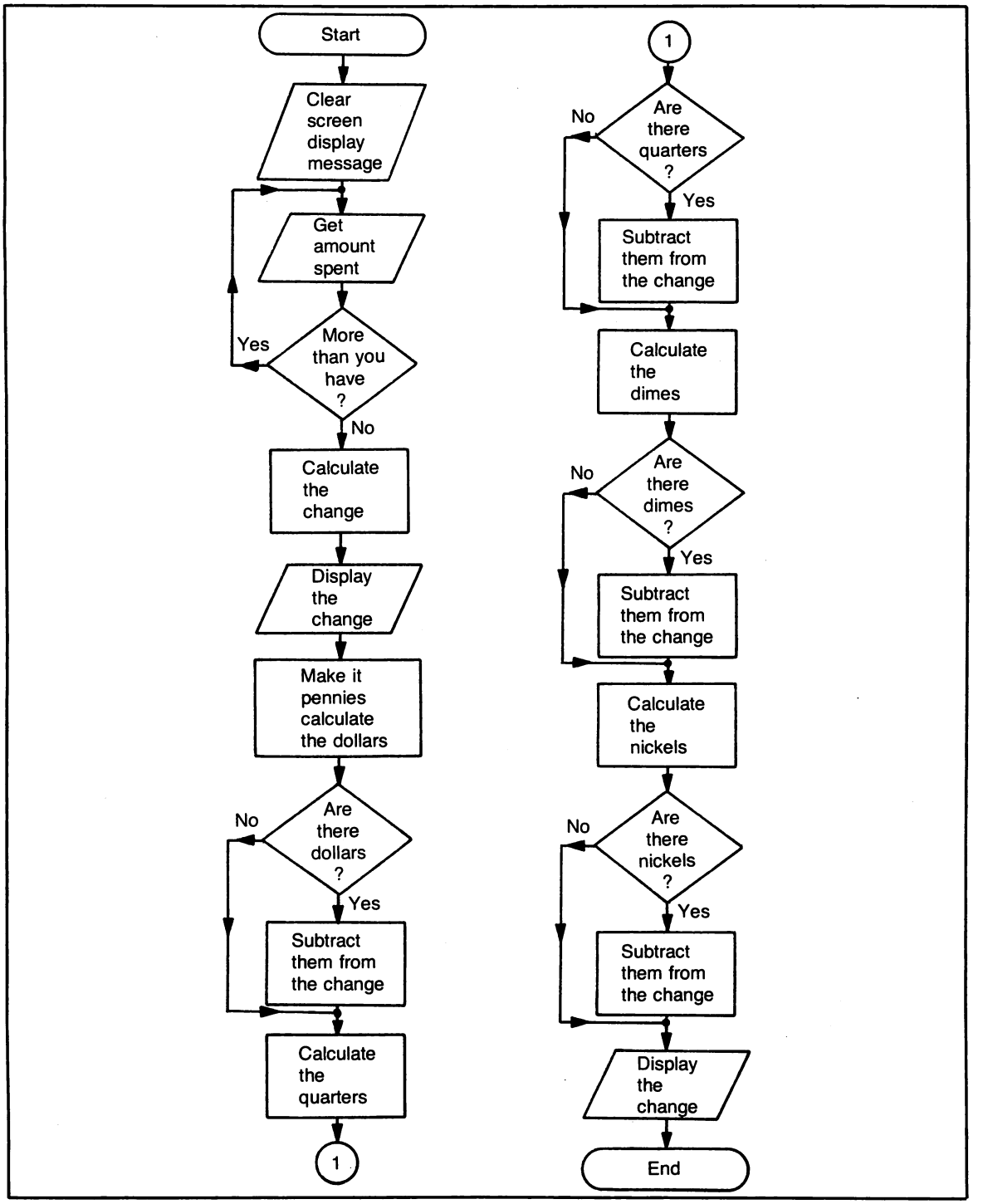


Fig. 13-1. Flowchart for Listing 13-1 Change.

Listing 13-1

```
100 REM LISTING 13-1
110 REM CHANGE
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: DISPLAY AT(5,9):"YOU H
AVE $5.00"
140 DISPLAY AT(7,1):"HOW MUCH DO YOU WAN
T TO" :: DISPLAY AT(9,1):"SPEND ?" :: AC
CEPT AT(9,9)BEEP VALIDATE(NUMERIC)SIZE(4
):SPEND
150 IF SPEND>5 THEN 140 ! DON'T SPEND MO
RE THAN YOU HAVE
160 CHANGE=5-SPEND ! GET AMOUNT LEFT
170 DISPLAY AT(11,5):"YOU HAVE $";CHANGE
 :: IF CHANGE*10=INT(CHANGE/.1)AND CHANG
E<>0 THEN DISPLAY AT(11,19):"0"
180 DISPLAY AT(11,21):"LEFT."
190 CHANGE=CHANGE*100 ! MAKE IT ALL PENN
IES
200 DOLLARS=INT(CHANGE/100)! GET THE NUM
BER OF DOLLARS
210 IF DOLLARS<>0 THEN CHANGE=CHANGE-DOL
LARS*100 ! REMOVE THE DOLLARS
220 QUARTERS=INT(CHANGE/25):: IF QUARTER
S<>0 THEN CHANGE=CHANGE-QUARTERS*25
230 DIMES=INT(CHANGE/10):: IF DIMES<>0 T
HEN CHANGE=CHANGE-DIMES*10
240 NICKEL=INT(CHANGE/5):: IF NICKEL<>0
THEN CHANGE=CHANGE-NICKEL*5
250 PENNIES=CHANGE
260 DISPLAY AT(13,1):DOLLARS;"DOLLAR(S)"
,QUARTERS;"QUARTER(S)" :: DISPLAY AT(15,
1):DIMES;"DIME(S)",NICKEL;"NICKEL(S)"
270 DISPLAY AT(17,1):PENNIES :: IF PENNI
ES=1 THEN DISPLAY AT(17,4):"PENNY" ELSE
DISPLAY AT(17,4):"PENNIES"
```

that you have, $5.00, is displayed on the screen.

Line 140 asks you how much money you want to spend. You can enter any amount from 5.00 to .00. Do not enter the dollar sign. The VALIDATE option will only allow you to enter numbers. The number that you enter will be stored in the SPEND variable.

Line 150 checks the amount entered. If it is greater than five, the computer will be sent back to line 140. You cannot spend more money than you have.

Line 160 subtracts the amount spent from five. This amount is stored in the CHANGE variable.

Line 170 shows you how much money you have left.

It checks to see if the amount left ends with a zero, like $1.20. If it does, we print another zero after the amount left. If we didn't, the computer would print $1.20 as $1.2. By multiplying the value of CHANGE by 10, we move the decimal one place to the right. For instance, 1.20 × 10 = 12 or 3.47 × 10 = 34.7. Next we divide the same number by .1. This also moves the decimal one place to the right, making 1.20/.1 = 12 or 3.47/.1 = 34.7. By taking the integer of the second number, we can check it to see if the cents ends with a zero. The integer of 12 is 12. Since the integer equals the real number, there was nothing after the decimal place. In the case of 34.7, the integer is 34. The two numbers are not equal, so there is something after the decimal point and no extra zero is required.

Line 180 prints the last word of the message.

Line 190 multiplies the amount stored in the CHANGE variable by 100. This converts the money into pennies.

Line 200 divides the value of CHANGE by 100 and places the integer value in the DOLLARS variable.

Line 210 tests the value of DOLLARS for a zero. If it is a zero than we have no dollars and computer will go on to the next line. If there are dollars, the computer must remove them from the value of CHANGE. Multiplying the value of DOLLARS by 100 and subtracting that amount from CHANGE removes the dollars from the change.

Line 220 finds the number of quarters in CHANGE by dividing the value of CHANGE by 25 and placing the integer value in the QUARTERS variable. The computer checks the value of QUARTERS. If it is not zero, the number of quarters are multiplied by 25 and subtracted from the amount in CHANGE.

Line 230 divides the amount left in CHANGE by 10 to find out how many dimes there are. The integer value of CHANGE divided by 10 is the number of dimes. The computer checks the value of DIMES to see if there are any dimes. If there are, the number of dimes are multiplied by 10 and subtracted from the value of CHANGE.

Line 240 finds the number of nickels by dividing the amount stored in CHANGE by five. If a nickel is removed, it is multiplied by five and subtracted from the amount in CHANGE.

Line 250 takes the amount left in CHANGE and stores it in PENNIES.

Lines 260-270 print the number of dollars, quarters, dimes, nickels, and pennies left. An IF . . . THEN . . . ELSE statement checks to see if the singular or plural version of penny should be used.

Notice that in every line that removed coins from the CHANGE variable, CHANGE was divided by the value of that coin. Then the computer was able to take the integer or whole number for the number of coins removed.

ABS

The ABSolute command gives the value of the number without the sign. The *absolute value* of negative three and the absolute value of positive three is the same—three. It is used when you need to know the difference between two numbers without regard to the sign. Listing 13-2 is a good example of how to use the absolute command. The program (flowcharted in Fig. 13-2) tells the user how many spaces from the hole the ball is without telling the user if the number should be greater or smaller.

Listing 13-2

Line 130 clears the screen and uses the randomize command so that a different number will be chosen by the computer each time the program is run.

Line 150 creates three graphics characters—the holes, the ball, and the ground (See Fig. 13-3). These characters are used in the program.

Line 160 uses the new character, number 130, to draw the ground. The RPT$ command tells the computer to print this character 28 times.

Line 170 draws the holes. The RPT$ command is used again. This time the pattern that is repeated is character number 128 and two spaces. This three-character pattern is printed on the screen ten times.

Line 190 numbers the holes. The computer steps through the FOR . . . NEXT loop by three's. This places the correct number under each hole.

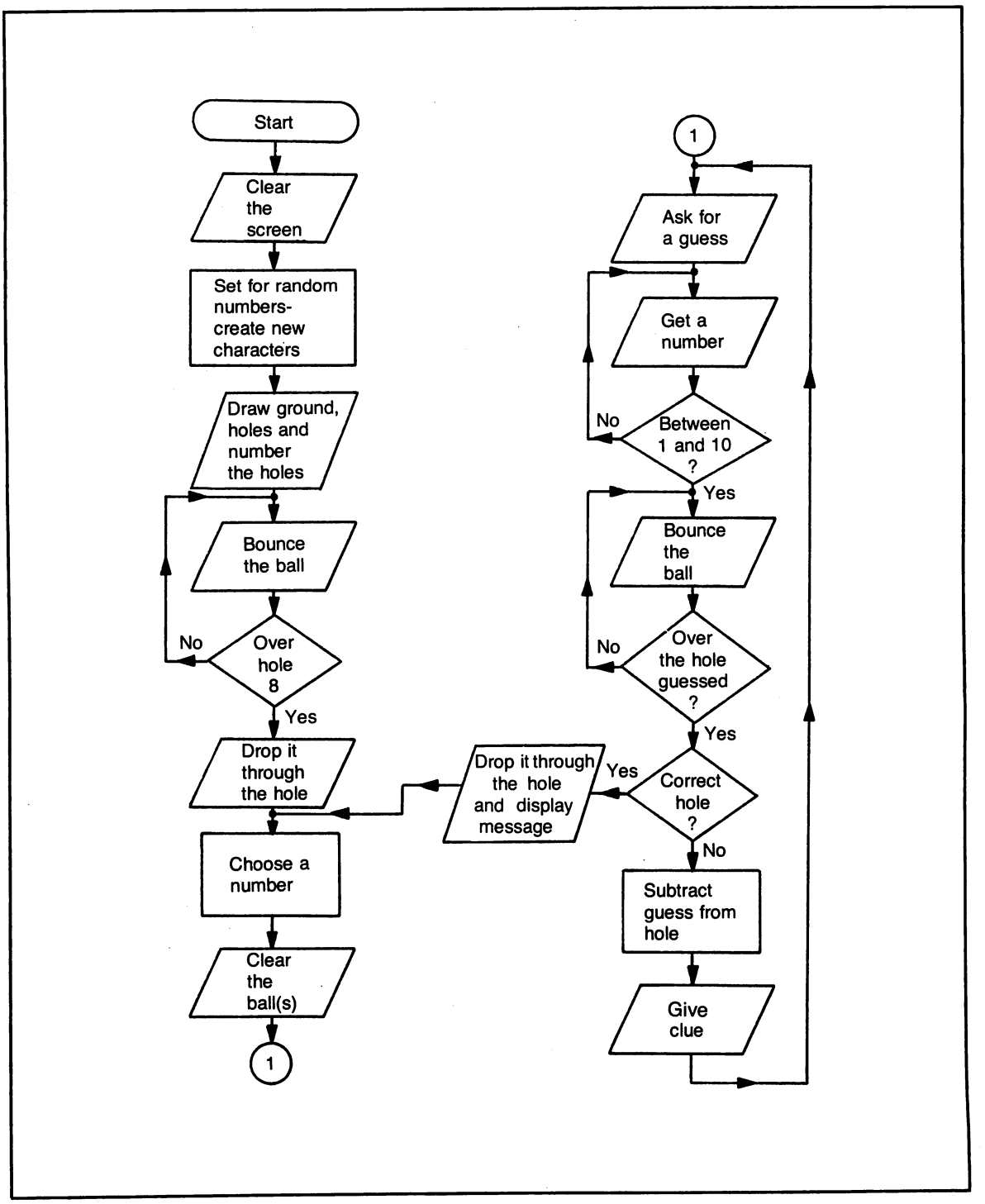


Fig. 13-2. Flowchart for Listing 13-2 Bounce.

Listing 13-2

```
100 REM LISTING 13-2
110 REM BOUNCE
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: RANDOMIZE
140 REM CREATE SOME GRAPHIC CHARACTERS
150 CALL CHAR(128,"8181818181818181")::
CALL CHAR(129,"00387C7C7C7C7C38"):: CALL
 CHAR(130,"00000000000000FF")
160 DISPLAY AT(10,1):RPT$(CHR$(130),28)!
 DRAW THE GROUND
170 DISPLAY AT(11,1):RPT$(CHR$(128)&"  "
,10)! DRAW THE HOLES
180 REM NUMBER THEM
190 COUNT=1 :: FOR COL=1 TO 25 STEP 3 ::
 DISPLAY AT(12,COL):STR$(COUNT):: COUNT=
COUNT+1 :: NEXT COL :: DISPLAY AT(12,27)
:STR$(COUNT)
200 HOLE=8 ! SAMPLE MAGIC HOLE
210 REM BALL DOWN
220 COUNT=1 :: FOR COL=1 TO 28 STEP 3 ::
 DISPLAY AT(9,COL):CHR$(129):: GOSUB 480
 :: DISPLAY AT(9,COL):" " :: IF COUNT=HO
LE THEN 250
230 DISPLAY AT(5,COL+2):CHR$(129):: GOSU
B 480 ! BOUNCING UP
240 DISPLAY AT(5,COL+2):" " :: COUNT=COU
NT+1 :: NEXT COL ! COUNT THE HOLE IT IS
OVER
250 DISPLAY AT(10,COL)SIZE(-1):CHR$(129)
:: GOSUB 480 :: DISPLAY AT(10,COL)SIZE(-
1):" " ! DROP IT IN
260 DISPLAY AT(10,COL)SIZE(-1):CHR$(130)
! COVER IT OVER
270 REM NOW START TO PLAY
280 HOLE=INT(RND*10)+1 ! THINK OF THE MA
GIC HOLE
290 REM LET HUMAN GUESS WHICH HOLE WILL
OPEN
300 DISPLAY AT(9,1):" " ! CLEAR ALL BALL
S
310 DISPLAY AT(15,1):"THE BALL CAN ONLY
FALL":"THROUGH ONE OF THESE HOLES. GUESS
```

continued on page 110

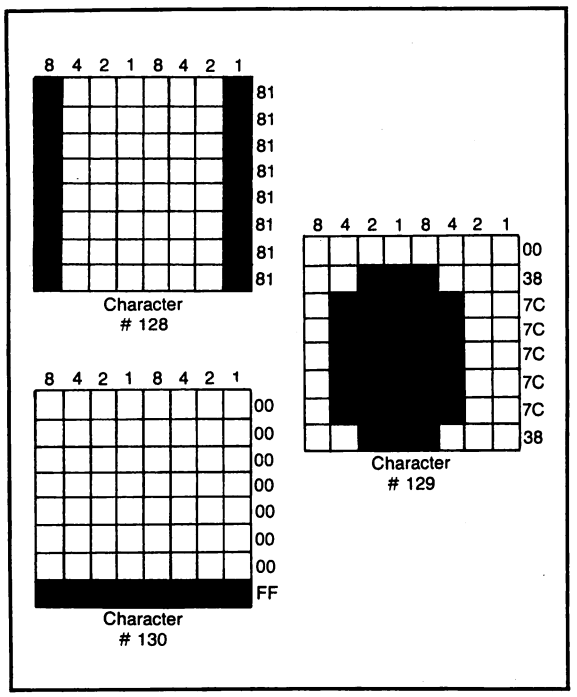


Fig. 13-3. Characters used in Bounce program.

Line 200 sets the HOLE variable to eight. When the example is run, the ball will disappear through hole eight.

Line 220 begins the FOR . . . NEXT loop that bounces the ball across the screen. The ball is character 129. It is printed on the screen. Then the computer goes to the subroutine at line 480. This is a timing loop to give you time to watch the ball. The ball is erased. Then the value of COUNT is compared to the value of HOLE. If both variables are the same, the ball is over the hole and the computer goes to line 250. If they are not the same, the computer continues with the next line.

Line 230 prints the ball four rows higher and two columns to the right. This is the up position. The computer uses the same subroutine at line 480 as a timing loop.

Line 240 erases the ball and adds one to COUNT. Now, COUNT contains the number of the next hole. The FOR . . . NEXT loop continues until the ball drops through a hole.

Line 250 drops the ball into the hole. The SIZE option is required in this line. If it is not used, the entire line after the ball is erased from the screen. The ball is printed on the same line as the ground. The computer uses the timing loop, then prints a space over that location. This gives the illusion that the ball fell into the hole.

Line 260 prints the character for the ground over the space. Again the SIZE option must be used to keep the rest of the line intact.

Line 280 begins the game. The computer chooses a number from one to ten. This is one of the holes on the screen.

Line 300 prints a space on the screen. Since there is no SIZE option with this DISPLAY AT command, the entire line will be cleared.

Line 310 prints the directions on the screen. The computer beeps, and waits for a number to be entered. The VALIDATE option will only accept one digit. This digit will be stored in the GUESS variable.

Line 330 checks the number entered. There are ten holes on the screen. If the number entered is less than zero or greater than ten, the computer will be sent back to line 310 to wait for another number.

Line 350 begins a FOR . . . NEXT loop to bounce the ball on the screen. The COUNT variable is set to one for the first hole. This variable keeps track of which hole the ball is over. The FOR . . . NEXT loop steps by threes so that the ball will always be printed over a hole. First the ball is printed just over the ground and above the hole. The computer uses the timing loop in line 480 to hold it there for a few seconds. Then the computer compares the value of GUESS with the value of HOLE. If both are the same, the computer will go on to line 380 to drop the ball through the hole. If the values are not the same, the computer will continue with the next program line.

Line 360 erases the ball. The SIZE option makes sure that only the ball is erased from that line. The ball is then printed four lines higher and two columns to the right. The computer uses the subroutine at line 480 to delay the bounce.

Line 370 erases the ball and adds one to COUNT. COUNT keeps track of which hole the ball is over. The loop continues.

Line 380 compares the value of GUESS with the

```
 WHICH ONE ?" :: ACCEPT AT(17,19)BEEP VA
LIDATE(DIGIT):GUESS
320 REM KEEP IT LEGAL
330 IF GUESS<1 OR GUESS>10 THEN 310
340 REM BOUNCE TO THAT HOLE
350 COUNT=1 :: FOR COL=1 TO 28 STEP 3 ::
 DISPLAY AT(9,COL)SIZE(-1):CHR$(129):: G
OSUB 480 :: IF COUNT=GUESS THEN 380 ! BA
LL DOWN
360 DISPLAY AT(9,COL)SIZE(-1):" " :: DIS
PLAY AT(5,COL+2):CHR$(129):: GOSUB 480 !
 BOUNCING UP
370 DISPLAY AT(5,COL+2):" " :: COUNT=COU
NT+1 :: NEXT COL ! COUNT THE HOLE IT IS
OVER
380 IF GUESS<>HOLE THEN 460 ! CHECK THE
GUESS - TRY AGAIN IF IT IS NOT RIGHT
390 DISPLAY AT(9,COL)SIZE(1):" "
400 DISPLAY AT(10,COL)SIZE(-1):CHR$(129)
:: GOSUB 480 :: DISPLAY AT(10,COL)SIZE(-
1):" " ! DROP IT IN
410 DISPLAY AT(10,COL)SIZE(-1):CHR$(130)
! COVER IT OVER
420 DISPLAY AT(20,1):" " ! CLEAR THE ENT
IRE LINE BEFORE PRINTING THE MESSAGE
430 FOR REPEAT=1 TO 5 :: DISPLAY AT(20,9
):"YOU GOT IT" :: GOSUB 480 :: DISPLAY A
T(20,9):" " :: GOSUB 480 :: NEXT REPEAT
440 GOTO 280 ! GO THINK OF ANOTHER NUMBE
R
450 REM WRONG GUESS - GIVE CLUE
460 DEVIATION=ABS(HOLE-GUESS)! SUBTRACT
THE GUESS FROM THE HOLE
470 DISPLAY AT(20,3):"YOU ARE";DEVIATION
;"SPACE(S) AWAY" :: GOTO 310 ! TELL HUMA
N HOW FAR AWAY - BUT NOT WHICH WAY
480 FOR DELAY=1 TO 25 :: NEXT DELAY :: R
ETURN ! LEAVE IT ON THE SCREEN
```

value of HOLE. If they are not the same, the computer will go on to line 460 to get a clue. If the two variables are the same, the computer will continue with the next program line.

Line 390 erases the ball from the screen.

Line 400 prints the ball on the same line as the ground. The SIZE command keeps the rest of the ground intact. The ball is erased again.

Line 410 prints the ground back over the hole and the ball has disappeared.

Line 420 clears the entire line before printing the message.

Line 430 is a FOR . . . NEXT loop that prints YOU GOT IT on the screen five times.

Line 440 sends the computer back to line 280 for another game.

Line 460 uses the ABS (absolute) command to find the difference between the number that the computer chose and the number entered. The value of GUESS is subtracted from the value of HOLE. The answer is stored in the DEVIATION. Because of the ABS command, the value of the DEVIATION will always be a positive number whether the difference between HOLE and GUESS is positive or negative.

Line 470 gives a clue to how far away from the ball the hole is. The number printed (the value of DEVIATION) gives the number of holes away from the correct hole the ball is, but does not give a hint about which direction. The program sends the computer back to line 310 for another guess.

Line 480 is the timing loop that is used in this program. It is a subroutine that keeps the ball on the screen for a few seconds.

SQU

The SQUare command finds the square root of a number or variable. The line below shows the syntax with SQU.

```
100  X=SQU(V):REM THE SQUARE ROOT OF
     'V' IS STORED IN 'X'
```

RND

RND is the most frequently used special function. So far, it has been used in every program in this book that requires any random numbers. Before the computer can choose a random number, the RANDOMIZE command must be used. Without it, the computer will chose the same numbers in the same order. With it, the computer will chose new numbers in a new order each time the program is run.

The number that the computer chooses is between zero and one. To obtain a whole number, the INTeger command is used with the random command. Try the program in Listing 13-3 with and without the RANDOMIZE command in line 140. Write down the numbers that the computer generates each time the program is run. Every time the program is run without the RANDOMIZE command, the number sequence is the same. With the command, a new set of numbers are generated each time the program is run. A flowchart of the program is shown in Fig. 13-4.

Listing 13-3

Line 130 clears the screen.

Line 140 contains the RANDOMIZE command. Try the program with and without this line.

Line 150 prints the example set that will be printed on the screen. This routine prints ten random numbers. These numbers will be between zero and one. They are all decimal values. There are no whole numbers. The FOR . . . NEXT loop will count from one to ten for the ten examples.

Line 160 chooses a random number. This number is a decimal.

Line 170 prints the number that the computer chose in line 160. This number is greater than zero but less than one.

Line 180 continues the loop.

Line 190 sends the computer to a subroutine that waits until a key is pressed. This gives you time to record the numbers that are on the screen.

Line 210 begins the loop that prints random numbers that are greater than zero. The FOR . . . NEXT loop will print ten numbers.

Line 220 chooses a random number. This time the random number is multiplied by five. Now the number chosen will be greater than zero but less than five.

Line 230 prints the number that was chosen. This number is greater than zero, but less than five. The numbers have decimals following the whole number.

Line 240 continues the loop.

Line 250 sends the computer to the subroutine at line 380.

Line 270 begins the loop that will print integers or

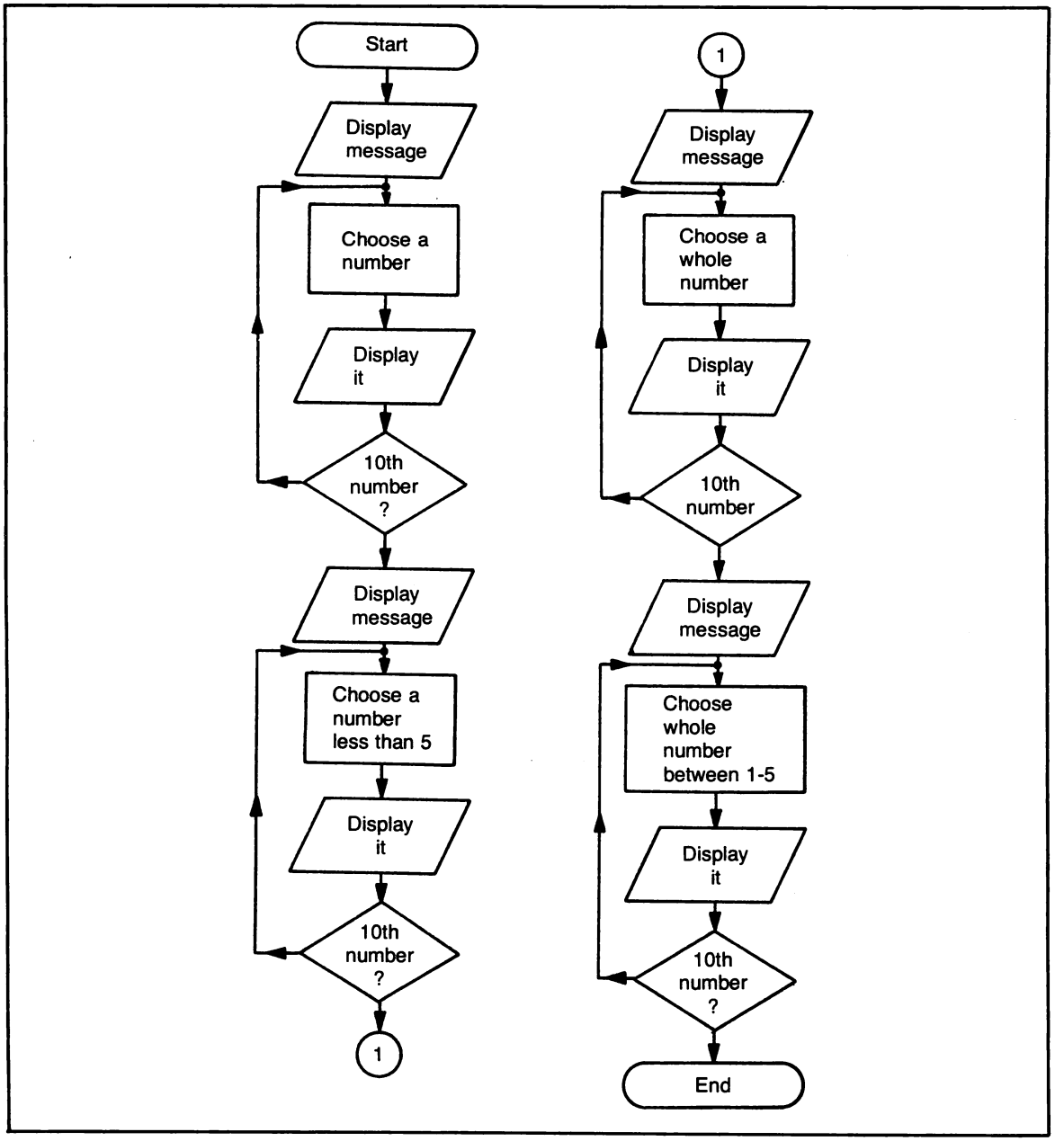


Fig. 13-4. Flowchart for Listing 13-3 Random.

whole numbers on the screen.

Line 280 multiplies that number chosen by five, but it also takes the integer (whole number) of the number chosen. The decimal portion of the number is disregarded.

Line 290 prints this number on the screen. The whole number can be a 0, 1, 2, 3, or 4.

Line 300 continues the loop.

Line 310 sends the computer to the subroutine that waits until a key is pressed.

Listing 13-3

```
100 REM LISTING 13-3
110 REM RANDOM
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR
140 RANDOMIZE
150 PRINT "RANDOM" :: FOR X=1 TO 10 ! GI
VE 10 SAMPLE NUMBERS
160 N=RND ! GET A RANDOM NUMBER
170 PRINT N,! SHOW IT
180 NEXT X
190 GOSUB 380
200 REM NOW SHOW IT WITH A MULTIPLE
210 PRINT :"MORE THAN 0" :: FOR X=1 TO 1
0
220 N=RND*5
230 PRINT N,! IT'S MORE THAN 0 BUT LESS
THAN 5
240 NEXT X
250 GOSUB 380
260 REM NOW WITH THE INTEGER COMMAND
270 PRINT :"INTEGER" :: FOR X=1 TO 10
280 N=INT(RND*5)! ANY NUMBER BETWEEN 0 A
ND 4
290 PRINT N,
300 NEXT X
310 GOSUB 380
320 REM NOW ADD 1
330 PRINT :"ADD 1 TO NUMBER" :: FOR X=1
TO 10
340 N=INT(RND*5)+1 ! ANY NUMBER BETWEEN
1 AND 5 INCLUSIVE
350 PRINT N,
360 NEXT X
370 END
380 PRINT :"PRESS ANY KEY TO CONTINUE"
390 CALL KEY(0,K,S):: IF S=0 THEN 390
400 RETURN
```

Line 330 prints random numbers that have one added to them. Adding one to the number ensures that zero will never be picked.

Line 340 chooses a random number, multiplies it by five, takes the integer of that number, then adds one to the number. This will give us a number from one to five inclusive. Always add one to the number chosen if you do not want zero as a possible number, but want the number multiplied by the random number as a possible number.

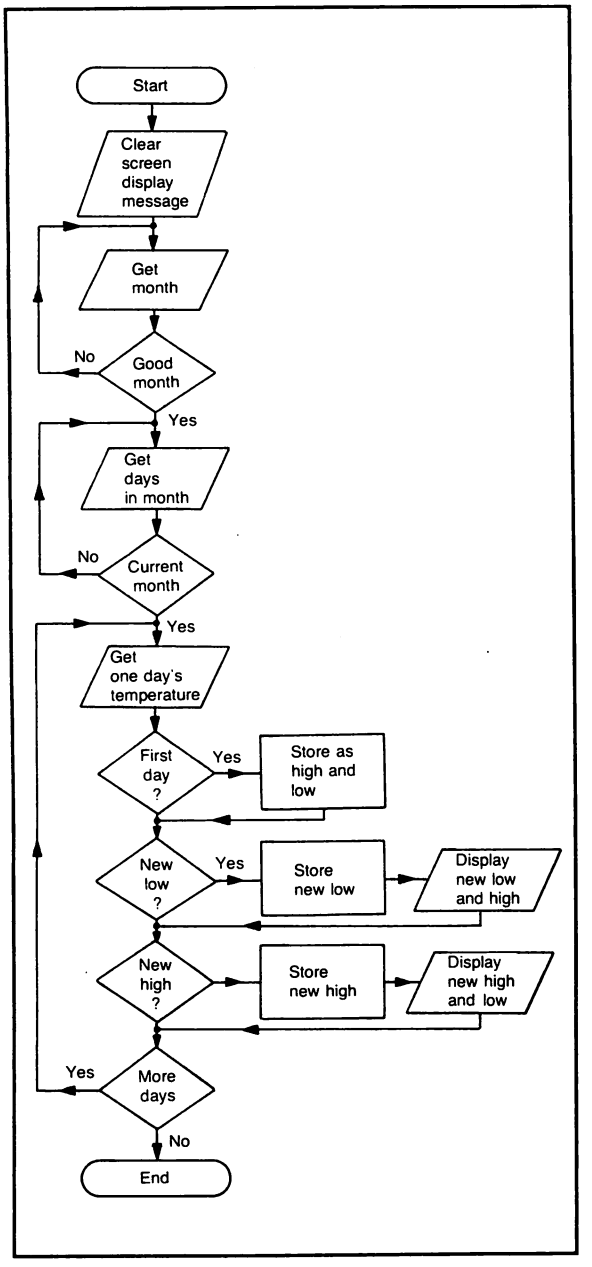


Fig. 13-5. Flowchart for Listing 13-4 Negatives.

Line 350 prints the number on the screen.
Line 360 continues the loop.
Line 370 ends the program, and separates the main program from the subroutine in line 380.
Line 380 prints the instructions on the screen. The program will wait until a key is pressed.
Line 390 uses the CALL KEY command to wait until a key is pressed. The S variable will not be zero when a key has been pressed. As long as S is zero, the computer will loop at this line.
Line 400 sends the computer back to the line that called it in the main program.

You can also specify which random pattern you want the computer to follow by placing a number or variable after the RANDOMIZE command.

RANDOMIZE (3)
RANDOMIZE (5)

The computer will follow the same pattern each time it encounters the same value. Try placing different values at line 140 and then record the results.

SGN

The SIGN command sets a variable to a negative one when the variable it checks is a negative number, a positive one if the variable is positive, and a 0 if the variable is zero. One use for this command is demonstrated in the temperature program in Listing 13-4 where we want to know if the temperature is above or below zero. (The program is flowcharted in Fig. 13-5.)

Listing 13-4

Line 130 clears the screen and asks for the number of the month. The VALIDATE option will only allow numbers. The SIZE option will not allow more than two digits to be entered. The number entered is stored in the MONTH variable.
Line 140 checks the number that was entered. If it is less than one or greater than 12, then it is not a month of the year, and the computer will go back to line 130 to wait for another entry.
Lines 150-160 is a FOR . . . NEXT loop that reads the number of days in the month entered. The data for this line is stored in line 290.
Line 180 checks to see if the month entered is February. If it is, then the program asks for the year. Only four digits will be accepted and stored in the YEAR variable.

Listing 13-4

```
100 REM LISTING 13-4
110 REM NEGATIVES
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DISPLAY AT(2,1)ERASE ALL:"ENTER THE
NUMBER OF THE":"MONTH ?" :: ACCEPT AT(3,
9)BEEP VALIDATE(DIGIT)SIZE(2):MONTH
140 IF MONTH<1 OR MONTH>12 THEN 130
150 FOR COUNT=1 TO MONTH :: READ DAYS !
GET THE NUMBER OF DAYS IN THE MONTH
160 NEXT COUNT
170 REM IF FEBRUARY THEN CHECK FOR LEAP
YEAR
180 IF MONTH=2 THEN DISPLAY AT(4,1):"ENT
ER YEAR ?" :: ACCEPT AT(4,14)BEEP VALIDA
TE(DIGIT)SIZE(4):YEAR
190 REM IF LEAP YEAR ADD 1 DAY
200 IF MONTH=2 THEN IF YEAR/4=INT(YEAR/4
)THEN DAYS=DAYS+1
210 FOR COUNT=1 TO DAYS
220 DISPLAY AT(10,1):"WHAT WAS THE TEMPE
RATURE":"FOR "&STR$(MONTH)&"-"&STR$(COUN
T)&"?" :: ACCEPT AT(11,12)BEEP VALIDATE(
NUMERIC)SIZE(3):TEMP
230 IF COUNT=1 THEN LOW=TEMP :: HIGH=TEM
P ! FIRST DAY SETS THE RECORDS
240 IF SGN(TEMP-LOW)=-1 THEN DISPLAY AT(
20,3):"NEW LOW TEMPERATURE: "&TEMP :: LO
W=TEMP :: DISPLAY AT(22,7):"HIGH TEMPERA
TURE: "&HIGH
250 IF SGN(HIGH-TEMP)=-1 THEN DISPLAY AT
(22,3):"NEW HIGH TEMPERATURE: "&TEMP ::
HIGH=TEMP :: DISPLAY AT(20,7):"LOW TEMPE
RATURE: "&LOW
260 NEXT COUNT
270 END
280 REM DAYS IN EACH MONTH
290 DATA 31,28,31,30,31,30,31,31,30,31,3
0,31
```

Line 200 divides the year entered by four and compares it to the integer of the year divided by four if the month entered is February. If the product of the year divided by four is the same as the integer of product, then it is a leap year and one is added to the number of days in the month of February.

Line 210 begins the FOR . . . NEXT loop that asks for the temperature for each day of the month.

Line 220 asks for the temperature of each day of the month. The number entered is stored in the TEMP variable.

Line 230 checks if this is the first day of the month. If so, both LOW temperature and HIGh temperature are set to the temperature entered.

Line 240 uses the SGN command to see if the temperature just entered is less than the current low temperature. If the value of TEMP is less than the value of LOW, the SGN of the numbers will be negative and there is a new low temperature. This temperature is printed on the screen. Low is now equal to TEMP. The high temperature is also printed.

Line 250 checks to see if the temperature entered is greater than the current high temperature. If the value of HIGH is less than the value of TEMP, the SGN of the numbers will be negative. The value of TEMP is a new high temperature. This value is printed on the screen and the value of TEMP is stored in HIGH. The low temperature is also printed on the screen.

Line 260 continues the loop until the temperature for each day of the month has been entered.

Line 270 contains an END statement. This stops the program before the data line.

Line 290 contains the data for the number of days in each month of the year.

Chapter 14
Working with Strings

When you store information in strings, you have easy access to the information. You can move it around in the string, use only part of it, or convert numbers that are in the string into numeric variables. By using strings you can easily manipulate the information any way you would like to.

ADDING STRINGS

One string can be added to another string. This type of addition is not the same as adding two numbers together. It is more like adding links onto a chain. Adding two or more strings together is called *concatenation.* An example of adding two strings is:

```
10  REM MAKE TWO STRINGS INTO ONE
20  A$="HELLO ":B$="THERE"! BE SURE
    THERE IS A SPACE AFTER THE 'O' IN
    HELLO
30  C$=A$&B$
40  PRINT C$
```

The words or expressions can be those that fill a complete string, those in quotation marks, or those that comprise part of a string.

SPLITTING STRINGS

SEG$

Sometimes it is necessary to "split" a string or copy part of it for one reason or another. The SEG$ "removes" part of a string and puts that information into another string, or prints it on the screen. The information is not actually erased from the string, but merely copied into another string or onto the screen. The information still remains in the original string.

The computer needs to know where the information is in the string and how many characters will be removed. The format for SEG$ is A$= SEG$(B$2,5), where B$ is the string that we are getting the information from, two is the number of the first character in the string that will be placed in A$, and five represents the number of characters that will be copied into A$. Try the next few lines with your computer.

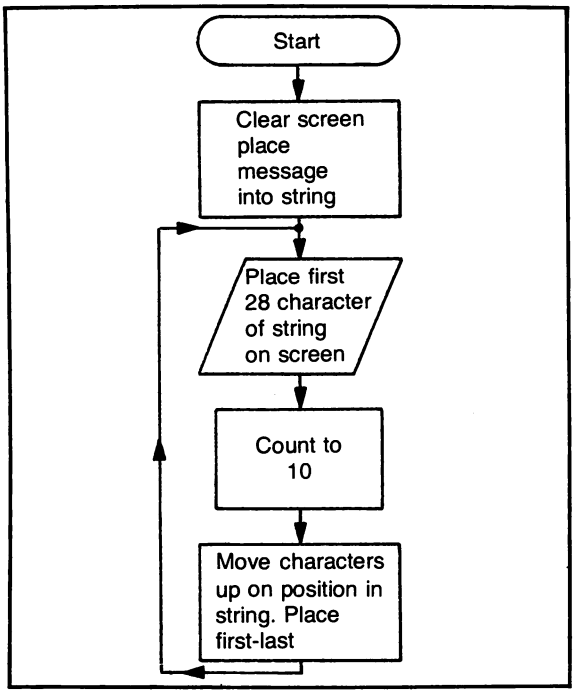


Fig. 14-1. Flowchart for Listing 14-1 Ticker Tape.

```
10  REM EXAMPLES OF SEG$
20  A$="The quick brown fox is really lazy"
30  WORD1$=SEG$(A$,1,3) ! THE WORD
    'THE'
40  WORD2$=SEG$(A$,4,6) ! QUICK WITH
    THE SPACE
50  WORD3$=SEG$(A$,10,20) ! LEAVE LAZY
60  WORD4$=SEG$(A$,30,5) ! GET 'LAZY'
70  B$=WORD1$&WORD4$&WORD3$&WORD
    2$
80  PRINT A$
90  PRINT B$
```

As you can see, the entire sentence can be changed by using the SEG$. In the program in Listing 14-1 (flowcharted in Fig. 14-1), we will create a ticker-tape effect across the screen by printing only a portion of the string at one time.

Listing 14-1

Line 130 clears the screen.
Line 150 places the message in A$. This message is 61 characters long.

Listing 14-1

```
100 REM LISTING 14-1
110 REM TICKER TAPE
120 REM L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR
140 REM STRING CONTAINS MESSAGE TO PRINT
150 A$=" The DOW JONES report at 12 noon
 is AT&T up 3 points ........."
160 REM PUT MESSAGE ON MIDDLE ROW STARTI
NG AT FIRST COLUMN---SCREEN ONLY 28 CHAR
ACTERS WIDE SO ONLY PRINT PART OF IT
170 DISPLAY AT(12,1):SEG$(A$,1,28)
180 REM TIMING LOOP...CHANGING THE 10 MA
KES IT PRINT FASTER OR SLOWER
190 FOR DELAY=1 TO 10 :: NEXT DELAY
200 REM MOVE FIRST CHARACTER OF STRING T
O END OF STRING WITH OTHER CHARACTERS MO
VING TOWARD FRONT OF STRING---GIVES STRI
NG WRAP-AROUND EFFECT
210 A$=SEG$(A$,2,LEN(A$)-1)&SEG$(A$,1,1)
220 GOTO 170 ! REPEAT UNTIL (CLEAR) IS P
RESSED
```

Line 170 prints the first 28 characters of the message on the screen.

Line 190 is the timing loop to give you a chance to read the message.

Line 210 uses the SEG$ command to move the first letter of the string into the last position. Taking all the characters of the string from the second position to the last. The amount of one must be subtracted from the length of A$ because we are not taking the first character. Next, the character that is in the first position of the string is added to the end of the string. Now all the characters have moved up one position.

Line 220 sends the computer back to line 170. The new string is printed on the screen.

By moving the characters in the string, but only printing those that occupy the first 28 positions, the letters move smoothly across the screen.

USING STRING FUNCTIONS

POS

In addition to adding strings together and rearranging them, we can search a string for a particular letter or character. The program in Listing 14-2 (flowcharted in Fig. 14-2) is an example of a telephone directory where the person's name, address, and telephone number are all stored in one string. The second part of the program searches for the character that separates the name from the address and phone number so that the information can be printed on the screen correctly.

Listing 14-2

Line 130 sets aside enough memory to hold 20 names and addresses.

Line 140 begins the loop that will enter the names.

Line 150 displays the instructions on the screen.

Line 160 gets the name and stores it in NAME$. It checks to see if the name is an XXX. If it is, then there are no more names to be entered. It leaves the loop and goes to line 240.

Lines 170-210 get the address, city, state, zip code, and phone number. Each of these entries are stored in a separate string variable.

Line 220 puts all the information in the various strings into one string. The COUNT variable indicates which element of the string array the name will be placed in. Each separate string is added to the others. A slash (/) is placed between each piece of information to separate each item from the next.

Line 230 continues the loop until 20 names have been entered.

Line 240 clears the string and asks for the name of the person whose address and/or phone number you want. The name will be stored in WANT$. The entire name must be entered just as it was typed in. If you misspelled the name when you entered it, you must misspell it now.

Line 250 checks the contents of WANT$. The program will stop if the string is empty (ENTER is pressed without typing any name).

Line 260 begins the FOR . . . NEXT loop that will search for the name that has been entered. We will count from the first name to the value of COUNT less one. We must stop one element before the value of COUNT, because COUNT is the value of XXX or 21. (Up to 20 names may be entered.) Counting all the way to the value of COUNT will cause the program to crash.

Line 270 sets the START variable to one. This variable will be used in the subroutine that begins with line 370. When the computer returns from this subroutine, it checks PART$ to see if it contains the same information as WANT$. If it does, then the name has been found and the computer proceeds to line 300. If the two string variables do not match, the computer will continue with the next program line.

Line 280 continues the loop until there are no more names in the list, or the match has been found. If no match has been found, the message will be displayed on the screen.

Line 290 is a delay loop to give you time to read the message. The computer is directed to line 240 to see if you want to look for another name.

Lines 300-310 stores the name, address, or other information that matches your inquiry. The same subroutine is used in line 370 to move this information from the element in DIRECTORY$ and place it in the correct string.

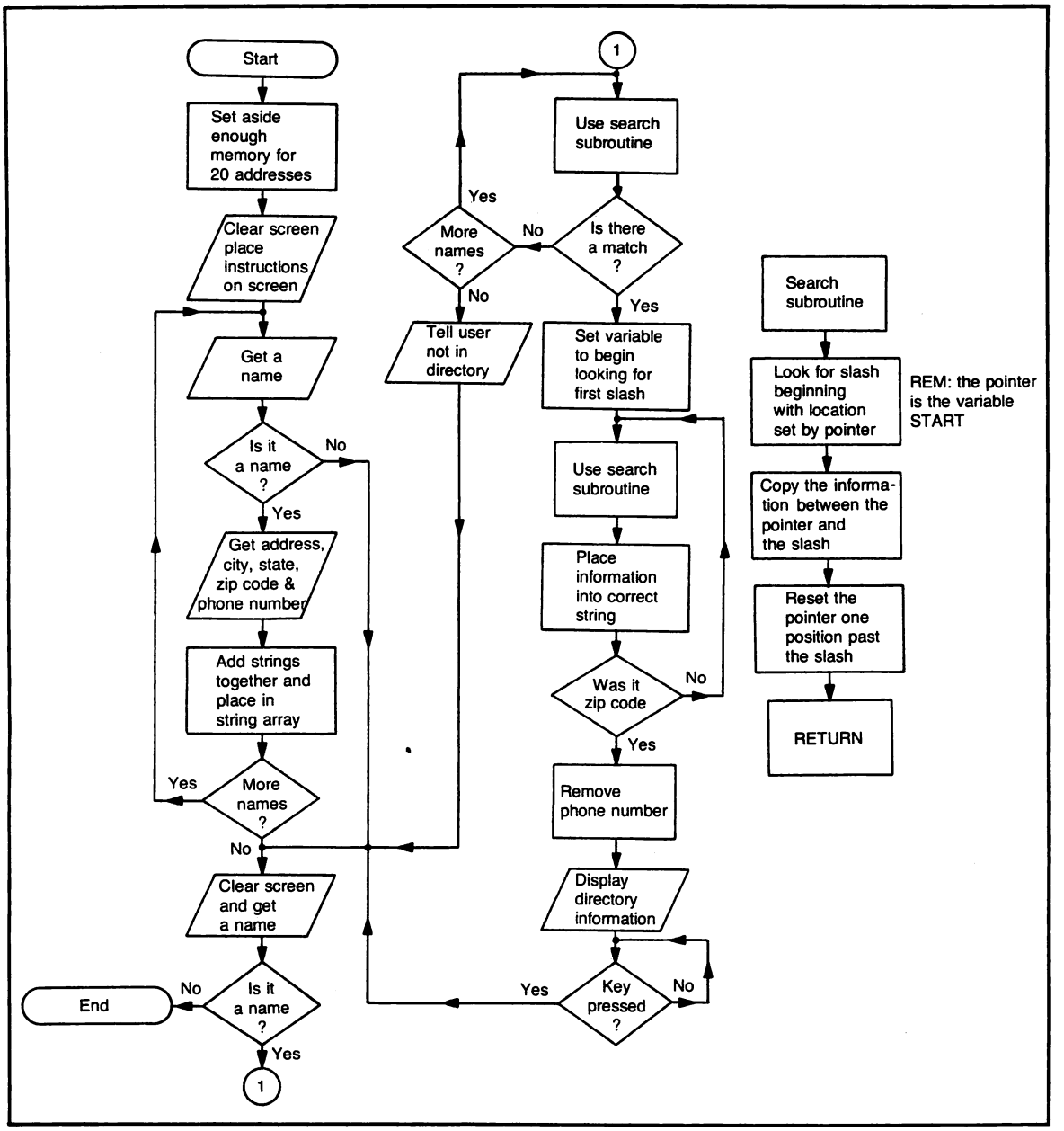


Fig. 14-2. Flowchart for Listing 14-2 Breaking Down Strings.

Line 320 finds out how long the DIRECTORY$ is for this particular element. The phone number is placed in PHONE$ by using the SEG$ command. The START variable will contain the first character after a slash. We will begin with this character and continue to the end of the string. We found out how long the string is and stored this value in L. By subtracting the position of the

Listing 14-2

```
100 REM LISTING 14-2
110 REM BREAKING DOWN STRINGS
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 DIM DIRECTORY$(20)
140 FOR COUNT=1 TO 20
150 DISPLAY AT(2,1)ERASE ALL:"ENTER THE
FOLLOWING OR 'XXX'"
160 DISPLAY AT(6,4):"NAME ?" :: ACCEPT A
T(6,10)BEEP:NAME$ :: IF NAME$="XXX" THEN
 240
170 DISPLAY AT(8,4):"ADDRESS ?" :: ACCEP
T AT(8,14)BEEP:ADDRESS$
180 DISPLAY AT(10,4):"CITY ?" :: ACCEPT
AT(10,11)BEEP:CITY$
190 DISPLAY AT(12,4):"STATE ?" :: ACCEPT
 AT(12,12)BEEP:STATE$
200 DISPLAY AT(14,4):"ZIPCODE ?" :: ACCE
PT AT(14,14)BEEP:ZIPCODE$
210 DISPLAY AT(16,4):"PHONE No. ?" :: AC
CEPT AT(16,16)BEEP:PHONE$
220 DIRECTORY$(COUNT)=NAME$&"/"&ADDRESS$
&"/"&CITY$&"/"&STATE$&"/"&ZIPCODE$&"/"&P
HONE$
230 NEXT COUNT
240 DISPLAY AT(22,1)ERASE ALL:"WHOSE ADD
RESS & PHONE DO YOUWANT?" :: ACCEPT AT(2
4,6)BEEP:WANT$
250 IF WANT$="" THEN STOP
260 FOR RECOUNT=1 TO COUNT-1
270 START=1 :: GOSUB 370 :: IF PART$=WAN
T$ THEN 300
280 NEXT RECOUNT :: DISPLAY AT(12,1)BEEP
:"NO ONE BY THAT NAME IS LIST-ED IN THIS
 DIRECTORY"
290 FOR DELAY=1 TO 1000 :: NEXT DELAY ::
 GOTO 240
300 START=1 :: GOSUB 370 :: NAME$=PART$
:: GOSUB 370 :: ADDRESS$=PART$ :: GOSUB
370 :: CITY$=PART$
310 GOSUB 370 :: STATE$=PART$ :: GOSUB 3
70 :: ZIPCODE$=PART$
320 L=LEN(DIRECTORY$(RECOUNT)):: PHONE$=
```

```
SEG$(DIRECTORY$(RECOUNT),START,L-START+1
)
330 DISPLAY AT(4,1)ERASE ALL:"NAME: ";NA
ME$ :: DISPLAY AT(6,1):"ADDRESS: ";ADDRE
SS$
340 DISPLAY AT(8,1):"CITY: ";CITY$ :: DI
SPLAY AT(10,1):"STATE: ";STATE$
350 DISPLAY AT(12,1):"ZIPCODE: ";ZIPCODE
$ :: DISPLAY AT(14,1):"PHONE No.: ";PHON
E$
360 DISPLAY AT(24,1):"PRESS ANY KEY TO C
ONTINUE" :: CALL KEY(0,KEY,STATUS):: IF
STATUS=0 THEN 360 ELSE 240
370 P=POS(DIRECTORY$(RECOUNT),"/",START)
380 PART$=SEG$(DIRECTORY$(RECOUNT),START
,P-START)
390 START=P+1
400 RETURN
```

character past the last slash and adding one, we know how many characters to move into PHONE$.

Lines 330-350 clear the screen, and then place the information from each string at the correct location on the screen. The information is taken from the DIRECTORY$.

Line 360 places a message on the screen and waits for a key to be pressed. The STATUS variable will be zero until a key has been pressed. The program will loop at this line until a key has been pressed. When a key is pressed, the computer will go to line 240 and wait for another name to be entered.

Line 370 begins the subroutine that first checks the string for the name that has been entered, then moves the information from the string into PART$ so that it can be stored in the correct string. The P variable will contain the location of the slash in the string. The computer will begin searching the string at the location specified by START. The first time this subroutine is used when we are looking for a name in the directory, START contains one. The computer begins with the first character of the string and continues until it finds the slash. The location of the slash is stored in P and the computer continues with the next line.

Line 380 takes the character whose position is that of START and places it just before the slash in PART$. When we were looking for the name, the P variable would contain the location of the first slash. Since START would be set to one, the computer would take all the characters beginning with the first and ending with the one just before the slash and store them all in PART$.

Line 390 adds one to P and stores the total in START. The next time the computer uses this subroutine, it will begin searching for the slash again, starting with the character just after the last slash it found. This is why we can use this subroutine to separate the name, address, and other elements in lines 300 to 310. The information between the slashes is placed in PART$ and transferred to the correct string in the main program.

Line 400 sends the computer back to the main program.

Look at Fig. 14-3. This is an example of a name stored in DIRECTORY$. The first time that the

computer uses the subroutine at line 370, the POS command will find the slash at location nine. The first time that this same subroutine is used to separate the parts of the string, START will be one, and P will be a nine. When we add one to P, START will become ten. The next time that the computer uses this subroutine, it will begin with the tenth position in the string, the one in 123 and search for the slash. The P variable will become 24 and all the characters from the one to the second "T" in street will be placed in the string variable PART$. The START variable will become 25 and be ready for the next search. This continues until all the parts of the string have been placed in each particular string for the name, address, city, and so on.

ASC

Every character that is printed on the screen has its own numeric code. Every letter, number, or character has a position in the character set. This position number is the ASCII value of that character. Most computers follow this code. This makes the systems somewhat compatible. For example, the ASCII value of the letter "A" is 65 on most systems. If the computer system does not use this code pattern, it cannot communicate with other systems. The graphics characters on different computers are usually different, but the numbers, symbols, and letters (both upper and lower case) all follow the standard ASCII codes. To find out what the code of a particular character is, simply type:

```
PRINT ASC("{character}")
```

Try these examples:

```
PRINT ASC("7")
PRINT ASC("Z")
PRINT ASC("/")
```

The following numbers should have appeared on your screen – 55, 90, 47. In the last program in this chapter, we will enter the ASCII values of these characters to place them on the screen.

CHR$

This command will place the character whose value is in the parentheses into a string, or print that character on the screen. Now that we know that each character has a particular value, we can use that value to print that character on the screen. Try these commands with your computer:

```
PRINT CHR$(33)
PRINT CHR$(71)
PRINT CHR$(61)
```

Your screen should display!, G, =. If you would like to see the entire character set displayed on your screen, type

```
FOR X=1 TO 256::PRINT X,CHR$(X)::NEXT X
```

You will notice that some of the codes do not have any characters. You can use these positions to create your own characters, or you can recreate any of the existing characters.

J	O	H	N	_	D	O	E	/	1	2	3	_	A	N	Y	_	S	T	R	E
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

E	T	/	N	E	W	_	T	O	W	N	/	S	T	/	9	9	9	9	9	/
22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42

8	0	2	-	4	5	3	-	2	3	0	0
43	44	45	46	47	48	49	50	51	52	53	54

Fig. 14-3. How the characters in a string are numbered.

VAL

The VALUE command was used in the guess program in a previous chapter. In that program, in the HURKY routine, the row and column was entered with one entry. This entry was stored in a string. There are times when you will want a number to be entered in a string. It may be that the number is used for the convenience of being able to accept two or more answers with one entry, or it may be because a string is the more acceptable way to handle the number for the purpose of the program. In any event, the actual value of the string can be determined by the VALUE command. The string is placed in parentheses following the command. The value of the string is placed in a NUMERIC variable or printed on the screen.

```
B$="1.23"::V=VAL(B$)::PRINT V
```

or

```
B$="1.23"::PRINT VAL(B$)
```

STR$

At other times, you may want to convert a variable into a string. This is accomplished by placing the value in parentheses. The string is placed in a string variable or printed on the screen.

```
B=2::B$=STR$(B)::PRINT B$
```

or

```
B=2::PRINT STR$(B)
```

You may remember that when you print a number on the screen, there is a space before and after the number. There may be times when you do not want this space after the number. By placing the value in a string, you eliminate this space. Note the difference between one of the two lines below and the next:

```
FOR X=1 TO 10::PRINT STR$(X);".”::NEXT X
```

```
FOR X=1 TO 10::PRINT X;".”::NEXT X
```

In the first example, the period will be printed immediately after the number. In the second, there will be a space between the number and the period.

RPT$

This command repeats a character a specific number of times, eliminating the need for duplicate typing. For example, instead of entering

```
20  B$="**********"
```

you could enter

```
20  B$=RPT$("*",20)
```

and the character in the quotation marks would be placed in B$ 20 times. The character or characters between the quotation marks can change, and the number of times the character(s) will be repeated can be changed, but the line cannot be longer than 255 characters. If more than one character is entered, the entire sequence will be repeated the number of times after the comma, as shown in the example below:

```
20 B$=RPT$("HELLO",8)
```

will produce

```
HELLOHELLOHELLOHELLOHELLOHELLO
HELLOHELLO
```

We have used this command in several programs in this book.

In the following program, we will use all the commands listed above. This program will create new characters that can be used in any other program (See Fig. 14-4.) These new characters can be stored on disk and entered into some other program later. The characters are displayed on the screen with the corresponding hex code for each line. If you are unfamiliar with binary to hex conversions, please turn to Appendix A. Then use the program listed in Listing 14-3 and flowcharted in Fig. 14-5.

NOTE: This program will run in a 16K cassette-based machine. Lines 1210 to 1260 are used for the

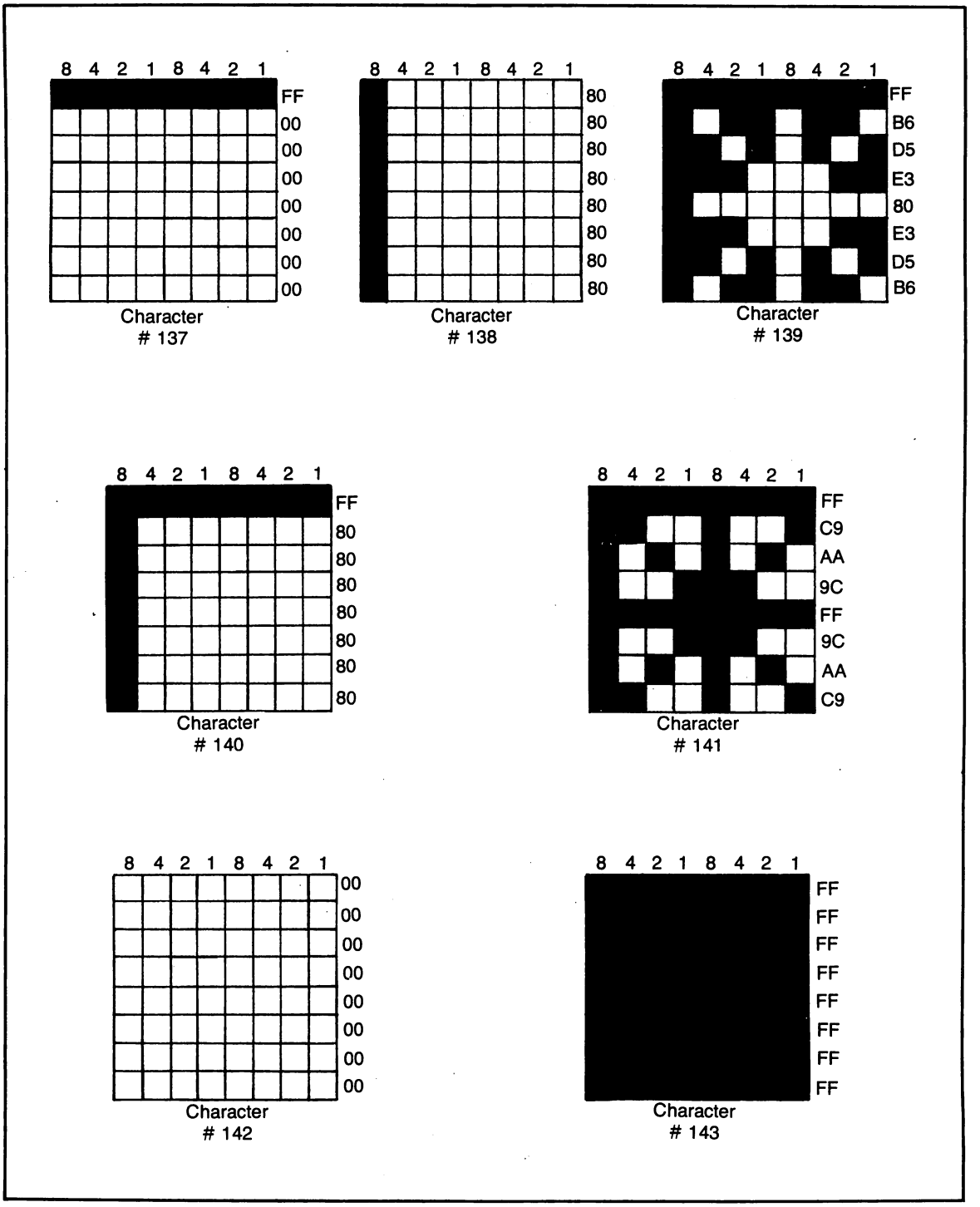


Fig. 14-4. Characters used for the grid and the cursor.

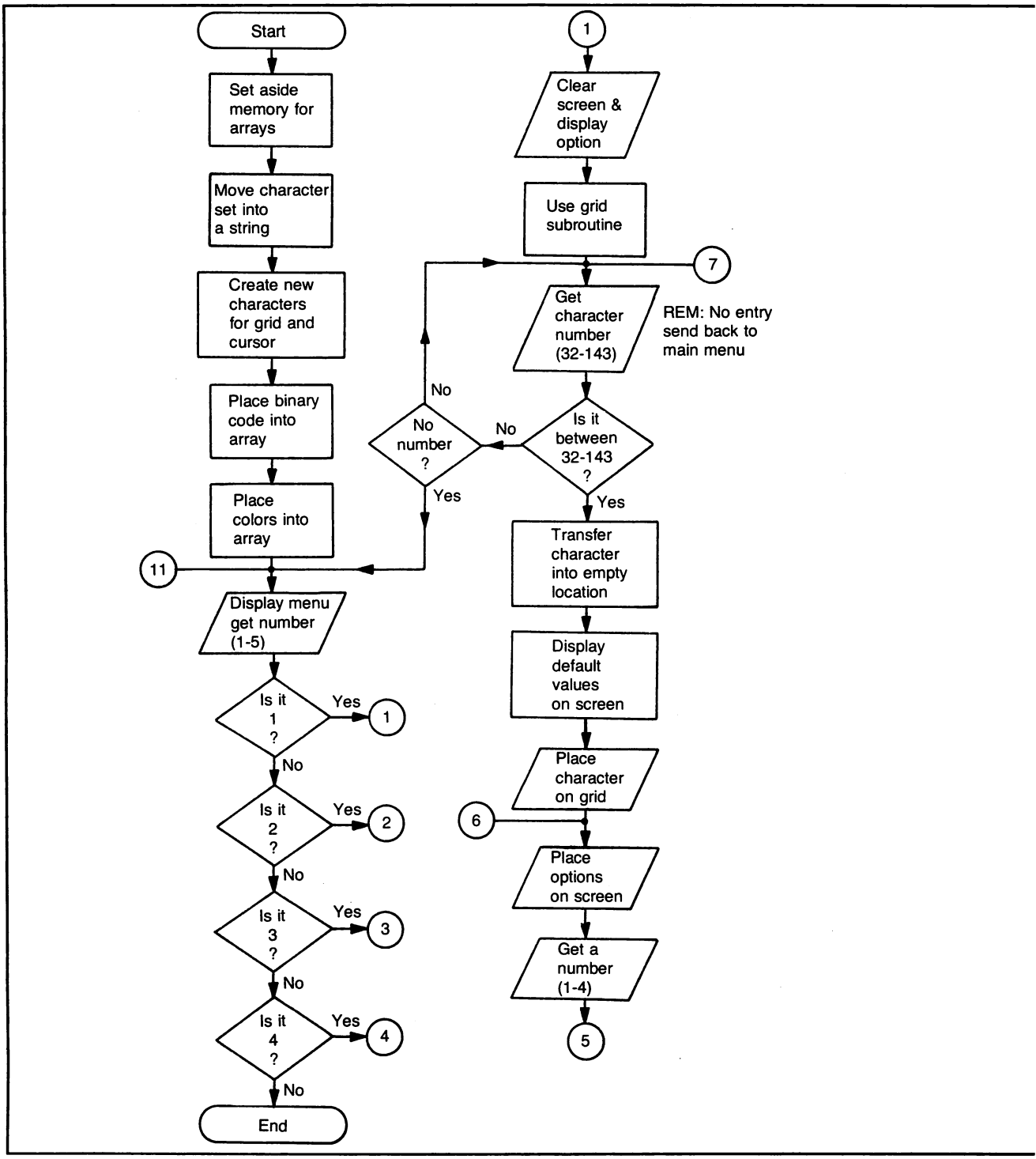
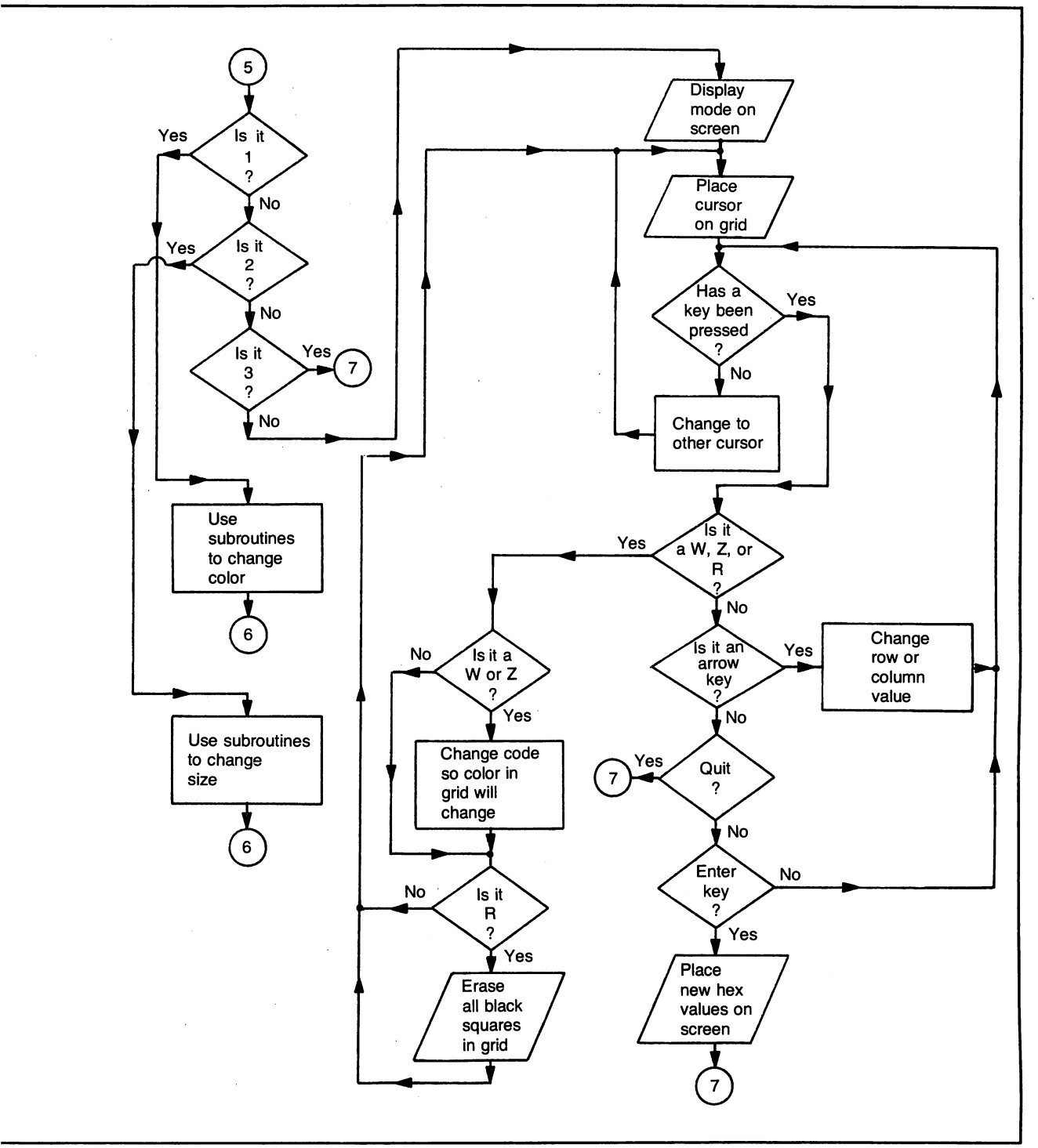


Fig. 14-5. Flowchart for Listing 14-3 Character patterns.


5
Is it 1 ?
Yes
No
Is it 2 ?
Yes
No
Is it 3 ?
Yes
7
No
Use subroutines to change color
6
Use subroutines to change size
6
Display mode on screen
Place cursor on grid
Has a key been pressed ?
Yes
No
Change to other cursor
Is it a W, Z, or R ?
Yes
No
Is it a W or Z ?
No
Yes
Change code so color in grid will change
Is it R ?
No
Yes
Erase all black squares in grid
Is it an arrow key ?
Yes
Change row or column value
No
Quit ?
Yes
7
No
Enter key ?
No
Yes
Place new hex values on screen
7

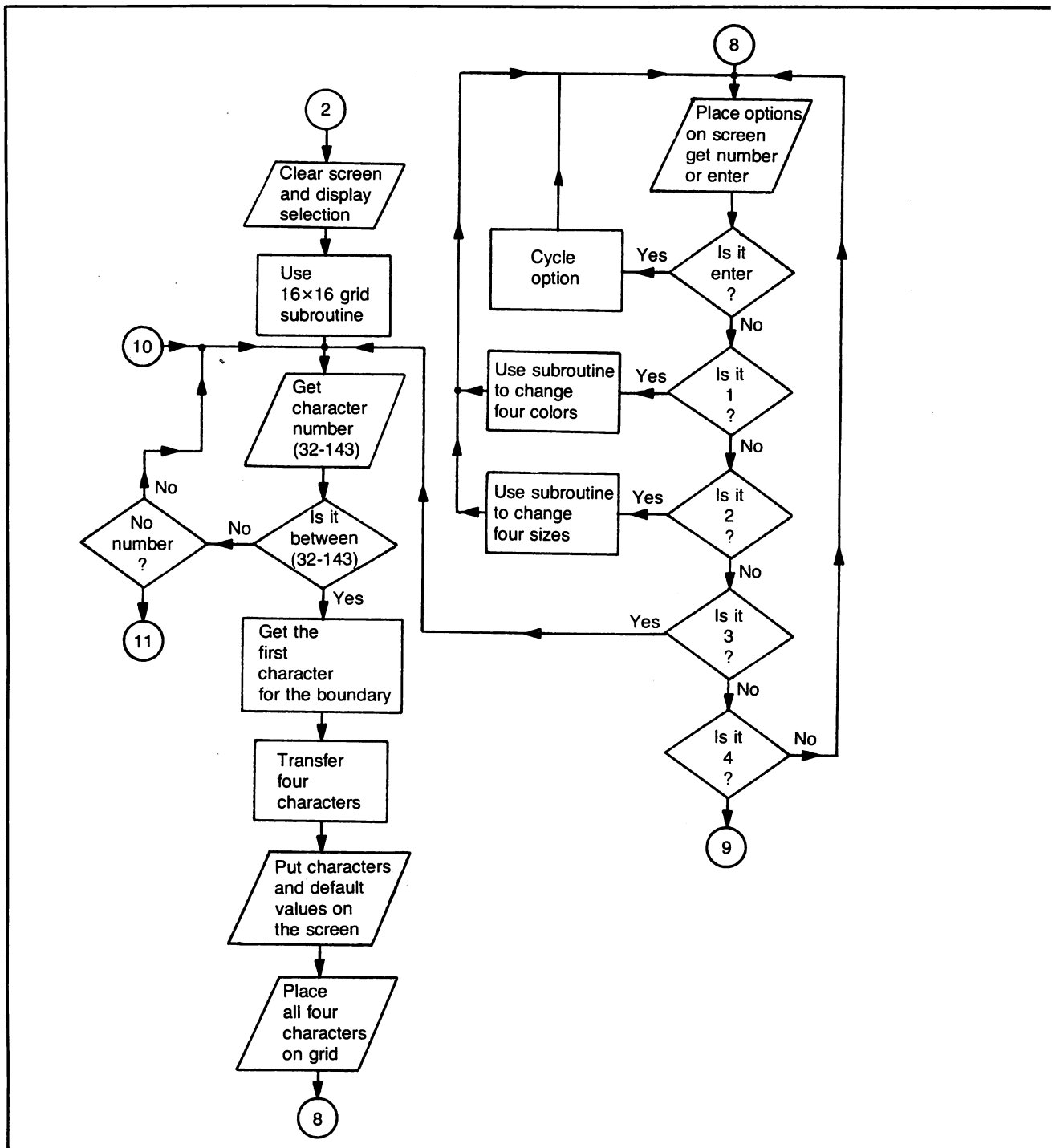
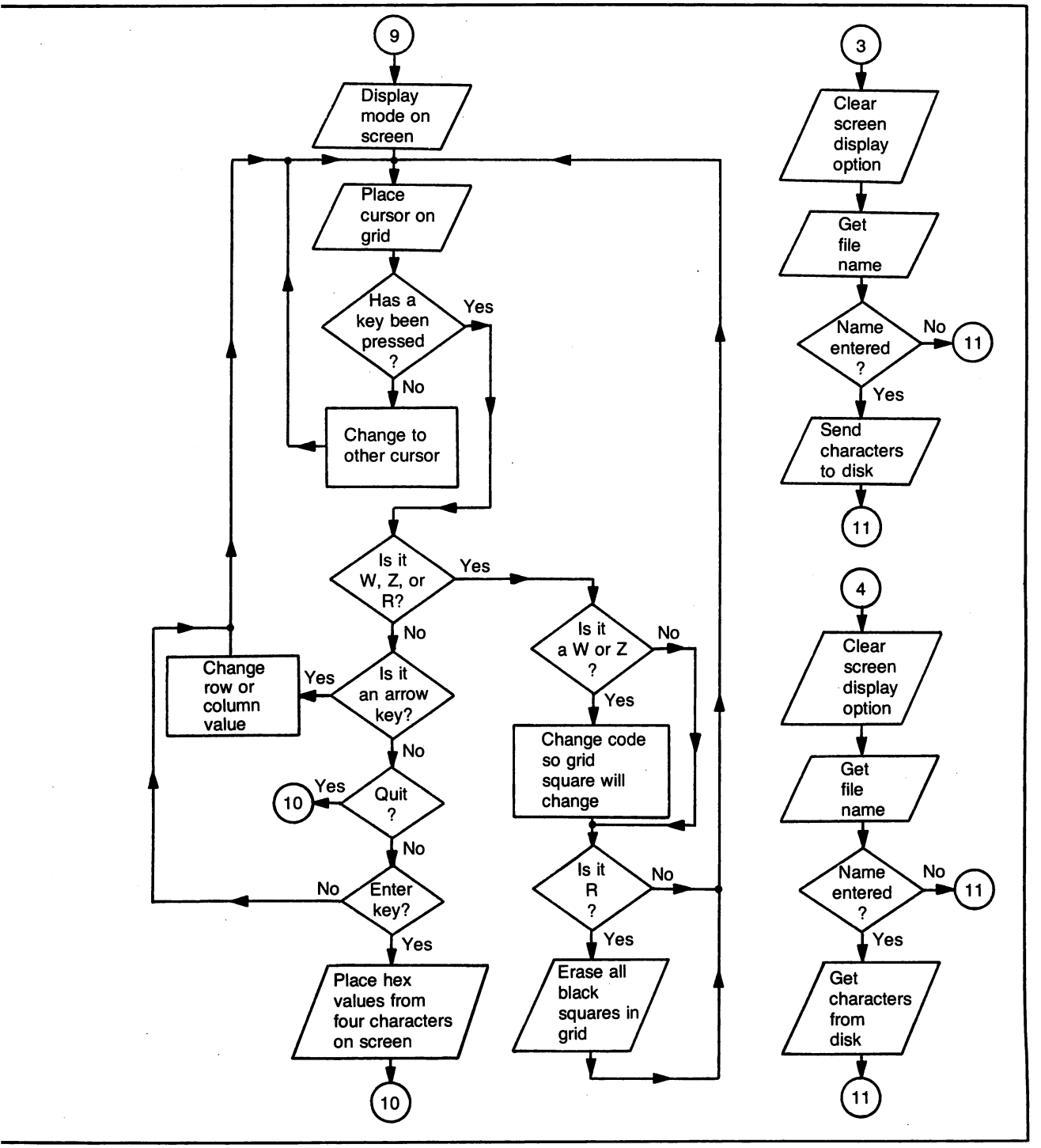


Fig. 14-5. Flowchart for Listing 14-3 Character patterns. (Continued from page 127.)


9
Display mode on screen
Place cursor on grid
Has a key been pressed ?
Yes
No
Change to other cursor
Is it W, Z, or R?
Yes
No
Is it a W or Z ?
No
Yes
Change code so grid square will change
Is it R ?
No
Yes
Erase all black squares in grid
Change row or column value
Yes
Is it an arrow key?
No
10
Yes
Quit ?
No
No
Enter key?
Yes
Place hex values from four characters on screen
10
3
Clear screen display option
Get file name
Name entered ?
No
11
Yes
Send characters to disk
11
4
Clear screen display option
Get file name
Name entered ?
No
11
Yes
Get characters from disk
11

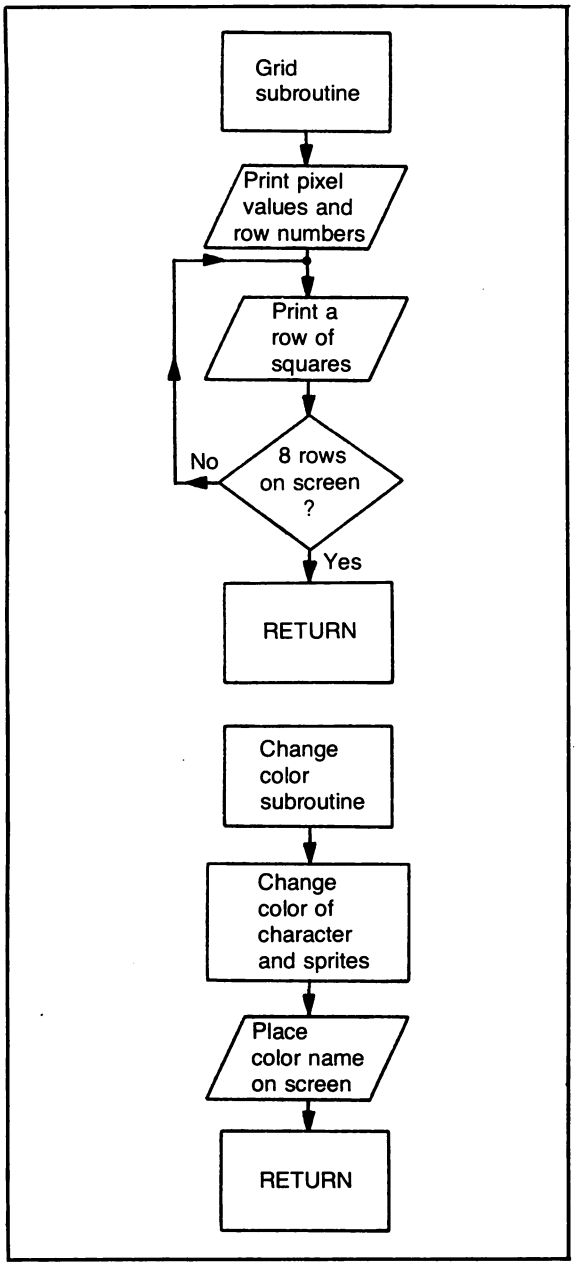


Fig. 14-5. Flowchart for Listing 14-3 Character patterns. (Continued from page 129.)

disk save and load. At the end of the program are the correct lines for cassette users. Do *not* enter any of the REM (remark) lines. The program will run without the remarks. Lines 1550 and 1560 are used only by the disk and should be omitted for the cassette version.

Listing 14-3

Line 130 sets aside the memory space for the characters that will be altered, the binary codes of the characters, and colors that are possible for these characters.

Line 140 reads the character patterns into CHAR$ array. The command CHARPAT gets the hex code for the character. This pattern is stored in CHAR$ so that we can alter them in this program. All the codes from the characters beginning with character 32 through character 143 are moved from the character set into this string. Any characters that you change will be changed in this string. Only when you save the new character set will the characters in the character set be changed. Each code is offset by 32 in the character set, but our array begins with the zero element of the array.

Line 150 reads the new character codes from the data in line 160. These coded are placed in characters 137 through 143 inclusive. These characters are changed in the character set that will be displayed on the screen for this program. These characters can, however, be used, changed, or modified for your character set.

Line 170 places all the possible hex codes in HEX$. These codes are in line 180 and will be used later in the program when we are converting the character codes into hex on the screen.

Line 190 reads the colors into COLR$ array. These color names will be displayed on the screen when we are changing the characters. The abbreviated color words are used when the larger characters are on the screen.

Lines 230-270 place the menu on the screen. This program will allow you to display a single character or a four-character pattern, load a character set, or save a character set. Use the single character option when you want to change only one character at a time. The four-character pattern is useful for creating sprites. In Chapter 16, we will discuss how four characters can be used together as one sprite. This option allows you to

Listing 14-3

```
100 REM LISTING 14-3
110 REM CHARACTER PATTERNS
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 DIM CHAR$(111),HEX$(15),COLR$(16,2)
140 FOR C=32 TO 143 :: CALL CHARPAT(C,CH
AR$(C-32)):: NEXT C
150 FOR C=137 TO 143 :: READ C$ :: CALL
CHAR(C,C$):: NEXT C
160 DATA FF,8080808080808,FFB6D5E380E3
D5B6,FF8080808080808,FFC9AA9CFF9CAAC9,0
0,FFFFFFFFFFFFFFFF
170 FOR C=0 TO 15 :: READ HEX$(C):: NEXT
 C
180 DATA 0000,0001,0010,0011,0100,0101,0
110,0111,1000,1001,1010,1011,1100,1101,1
110,1111
190 FOR N=1 TO 2 :: FOR C=1 TO 16 :: REA
D COLR$(C,N):: NEXT C :: NEXT N
200 DATA TRANSPARENT,BLACK,MEDIUM GREEN,
LIGHT GREEN,DARK BLUE,LIGHT BLUE,DARK RE
D,CYAN
210 DATA MEDIUM RED,LIGHT RED,DARK YELLO
W,LIGHT YELLOW,DARK GREEN,MAGENTA,GRAY,W
HITE
220 DATA TRAN,BLCK,MGRN,LGRN,DBLU,LBLU,D
RED,CYAN,MRED,LRED,DYEL,LYEL,DGRN,MGNT,G
RAY,WHIT
230 DISPLAY AT(2,3)ERASE ALL:"CHARACTER/
SPRITE PATTERNS"
240 DISPLAY AT(5,10):"MAIN MENU" :: DISP
LAY AT(10,2):"1. DISPLAY SINGLE PATTERN"
 :: DISPLAY AT(12,2):"2. DISPLAY FOUR PA
TTERNS"
250 DISPLAY AT(14,2):"3. LOAD CHARACTER
SET" :: DISPLAY AT(16,2):"4. SAVE CHARAC
TER SET"
260 DISPLAY AT(18,2):"5. EXIT PROGRAM"
270 DISPLAY AT(22,6):"ENTER CHOICE ?" ::
 ACCEPT AT(22,22)BEEP VALIDATE("12345")S
IZE(1):C :: ON C GOTO 310,720,1250,1220,
290
280 REM EXIT PROGRAM
290 END
```

continued on page 133

see the entire character the way it would be displayed on the screen. You can load into the computer any character set that you have saved, and, of course, you can save the character set that you are working on. When you are finished with the program, you can exit it with option 5, the EXIT PROGRAM. The VALIDATE option is set to accept only the numbers one through five inclusive. Your choice is stored in the C variable and the computer is sent to the correct routine.

Line 290 is the END command. When you choose options 5, the computer will be directed to this line and the program will end.

Line 310 begins option 1—display single pattern. The screen is cleared and the option chosen is displayed at the top.

Line 320 sets the RN variable to one. This variable will count the number of grid rows placed on the screen. The computer uses the subroutine beginning with line 1270 to draw the 8×8 grid on the screen. This character grid will be numbered with the values of each pixel.

Line 330 asks for the character number that you want displayed on the screen. The subroutine at line 1540 accepts the character number. The computer returns to this line with the number of the character that you would like displayed on the screen. If you do not enter a number, but simply press ENTER, the computer will erase any sprites that may have been created and send you back to the main menu. If you have entered a character number, the computer will use the VALUE command to get the number from the string and place this value into the W variable.

Line 340 checks the value of the number entered to make sure that it is a character between 32 and 143. If the character number entered cannot be changed with this program, the computer will be sent back to line 330 for another character number. Otherwise, the character number 135 will be changed to reflect the character that you want to change. This character is changed in the character set, not in the actual character. The character you create will then be printed on the screen so you can see what it looks like. If you have already altered the character, your new character will be displayed; otherwise, the original character will be shown.

Line 350 sets the CLR variable to two and M to 1. These are default values used in this program. The color of the character on the screen will be black and the size of the sprite will be normal. The labels for sprite and magnify will be printed on the screen.

Line 360 prints the label for the color on the screen, then uses the subroutine at line 1440. This routine sets the color and sprite, and places the character and color on the screen. The subroutine at line 1460 is called to set the size of the sprite.

Lines 370-400 take each byte of the character and place it on the screen on the large grid. First, the two character hex byte is removed from the CHAR$ at the location of the character that we want displayed. The subroutine at line 1300 is used to break down the hex code into binary. We must use binary because the characters are all binary codes. The pixel is either on or off. The logic statement IF BT will print a black square at the correct location of the grid if the value of BT is equal to one. The result is the same if we tell the computer IF BT or IF BT=1. The computer will continue with the program statement if the value of BT is not zero. If the value of BT is false, or zero, the computer will go on to line 390 and print an empty box on the screen. This loop continues until all eight hex codes for the character are displayed on the grid.

Lines 420-440 display new options on the screen. Now you can change the color of the character on the screen, change the magnification of the sprite, display a new character on the screen, or change the character that is on the screen. Your entry will be stored in the OP variable.

Line 450 uses the subroutine at line 1580 to clear the last five rows on the screen after the option number has been entered. The computer is sent to the correct routine based on the number entered.

Line 460 sends the computer to the subroutine at line 1600. Here you will be able to change the color of the character that is on the screen. Once the new color has been entered, the computer will

```
300 REM DISPLAY SINGLE CHARACTER/SPRITE
PATTERN
310 DISPLAY AT(2,4)ERASE ALL:"DISPLAY SI
NGLE PATTERN"
320 RN=1 :: GOSUB 1270
330 DISPLAY AT(6,24):"CHAR." :: GOSUB 15
40 :: IF W$="" THEN CALL DELSPRITE(#1)::
 GOTO 230 ELSE W=VAL(W$)
340 IF W<32 OR W>143 THEN 330 ELSE CALL
CHAR(135,CHAR$(W-32))
350 CLR=2 :: M=1 :: DISPLAY AT(10,23):"S
PRITE" :: DISPLAY AT(12,24):"MAG"
360 DISPLAY AT(17,23):"COLOR" :: GOSUB 1
440 :: GOSUB 1460
370 FOR R=1 TO 8 :: B$=SEG$(CHAR$(W-32),
R*2-1,2):: GOSUB 1300
380 FOR COL=1 TO 8 :: BT=VAL(SEG$(BN$,CO
L,1)):: IF BT THEN DISPLAY AT(R+7,COL+10
)SIZE(1):CHR$(143):: GOTO 400
390 DISPLAY AT(R+7,COL+10)SIZE(1):CHR$(1
40)
400 NEXT COL :: DISPLAY AT(R+7,20)SIZE(2
):B$ :: NEXT R
410 REM GET DISPLAY OPTIONS
420 DISPLAY AT(2,1):"   DISPLAY SINGLE P
ATTERN" :: GOSUB 1580
430 DISPLAY AT(20,1):"ENTER NUMBER WANTE
D ?" :: DISPLAY AT(21,5):"1. CHANGE COLO
R" :: DISPLAY AT(22,5):"2. CHANGE MAGNIF
ICATION"
440 DISPLAY AT(23,5):"3. DISPLAY NEW COD
E" :: DISPLAY AT(24,5):"4. ALTER CHARACT
ER" :: ACCEPT AT(20,23)BEEP VALIDATE("12
34")SIZE(1):OP
450 GOSUB 1580 :: ON OP GOTO 460,470,330
,490
460 GOSUB 1600 :: GOSUB 1440 :: GOTO 420
470 GOSUB 1630 :: GOSUB 1460 :: GOTO 420
480 REM ALTER SINGLE CHARACTER PATTERN
490 DISPLAY AT(2,3):"ALTER CHARACTER PAT
TERN"
500 CR,OLR=8 :: CC,OC=11 :: CUR$=CHR$(14
1)
```

continued on page 135

be sent to the subroutine at line 1440 to change the character's color. The computer will then be directed to line 410 and the menu will be displayed.

Line 470 is used to change the size of the sprite on the screen. Any character can be used as a sprite, and the sprite can be two different sizes. The way the sprite appears on the screen when the character is printed is the normal size. By entering a two for the size, you can enlarge the single sprite. The computer uses the subroutine at line 1630 to get the number of the size of sprite that you want. It then uses the subroutine at line 1460 to change the size of the sprite. Once the size of the sprite has been changed, the computer will display the menu on the screen.

Line 490 begins the routine to change the character that is on the screen. At the top of the screen, the option will be displayed.

Line 500 sets the CR and OLR variables to eight. These variables will keep track of the row the cursor is in on the screen. The CC and OC variables are set to 11. These variables keep track of the column of the cursor. Otherwise, the cursor could not be printed and moved around on the screen. The string variable CUR$ is set to character 141. This character has been predefined by the program and will indicate which pixel can be turned on or off.

Line 510 uses the GCHAR command to find out what character is on the screen at the location OLR, OC+2. The value of the character is stored in the T variable. We need to keep track of what is on the screen before we print the cursor there so that when we move the cursor, we can replace the original character in that location.

Line 520 prints the cursor on the screen. The CR and CC variables tell the computer where on the screen the cursor should be placed.

Line 530 uses the CALL KEY command to see if a key has been pressed. The status of the keyboard is stored in the S variable. If the value of S is not a zero, then a key has been pressed and the computer will go on to line 560. Otherwise, it will continue with the next program line.

Line 540 checks to see which character is being used for the cursor. There are two characters being used for the cursor. One is the negative image of the other, so if the character is 141, the computer will change it to 139, otherwise, it will make character 141 as the cursor.

Line 550 sends the computer to line 520, the line that prints the cursor on the screen. Until a key is pressed, the computer will continue this loop, changing the cursor back and forth between the positive and negative images. This makes the cursor flash on the screen.

Line 560 checks the value of the key pressed. If the Key value is 82, 87, or 90, then the R, W, or Z key has been pressed. The W key will blacken the square and the R key will erase all the squares. the computer will be sent to line 670 to perform the correct routine.

Line 570 prints the character that was originally in the square back in the square so that the cursor can move.

Lines 580-610 check the value of the KEY variable to see if one of the arrow keys have been pressed. The keys to move the cursor are E, X, D, and S. FCTN need not be pressed to move the cursor; simply pressing the correct arrow key will move the cursor in that direction. The computer checks for each value separately. If the value of K matches on the arrow keys, the OLR or OC variable will be changed accordingly. That is, its value will be increased by one if the cursor moves to the right or down, and is decreased by one if the cursor should move to the left or up. If the value of KEY is 81 the Q has been pressed and the computer will quit this routine by going to line 410. If the value of KEY is not 13, that is, the ENTER key has not been pressed, the values of OLR and OC are placed into the CR and CC variables. The computer is sent to line 510 to move the cursor in the correct direction.

Line 640 begins the routine that places the hex codes for the new character on the screen. Up until this time, the hex codes that were on the screen were for the original character. This routine changes them to set up the new character. P$ is cleared to hold the new codes for the new character. The computer will use the subroutine

```
510 CALL GCHAR(OLR,OC+2,T)
520 DISPLAY AT(CR,CC)SIZE(1):CUR$
530 CALL KEY(0,K,S):: IF S<>0 THEN 560
540 IF CUR$=CHR$(141)THEN CUR$=CHR$(139)
ELSE CUR$=CHR$(141)
550 GOTO 520
560 IF K=82 OR K=87 OR K=90 THEN 670
570 DISPLAY AT(OLR,OC)SIZE(1):CHR$(T)
580 IF K=69 THEN OLR=OLR-1 :: IF OLR=7 T
HEN OLR=15
590 IF K=88 THEN OLR=OLR+1 :: IF OLR=16
THEN OLR=8
600 IF K=68 THEN OC=OC+1 :: IF OC=19 THE
N OC=11
610 IF K=83 THEN OC=OC-1 :: IF OC=10 THE
N OC=18
620 IF K=81 THEN 420 ELSE IF K<>13 THEN
CR=OLR :: CC=OC :: GOTO 510
630 REM CONVERT NEW CHARACTER TO HEX-COD
E & INSERT IN CHAR$(W)
640 P$="" :: FOR OLR=8 TO 15 :: SC=11 ::
 EC=18 :: GOSUB 1350 :: P$=P$&B$
650 DISPLAY AT(OLR,20)SIZE(2):B$ :: NEXT
 OLR :: CHAR$(W-32)=P$ :: CALL CHAR(135,
P$):: GOTO 420
660 REM SET, CLEAR, OR ERASE
670 IF K=87 THEN T=143
680 IF K=90 THEN T=140
690 IF K<>82 THEN 520
700 RN=1 :: GOSUB 1270 :: DISPLAY AT(6,2
4):"CHAR." :: DISPLAY AT(8,26):CHR$(135)
:: GOTO 500
710 REM DISPLAY FOUR CHARACTER/SPRITE PA
TTERN
720 DISPLAY AT(2,4)ERASE ALL:"DISPLAY FO
UR PATTERNS"
730 GOSUB 1470
740 W$="CHAR" :: R=8 :: GOSUB 1510 :: GO
SUB 1540 :: IF W$="" THEN CALL DELSPRITE
(#1):: GOTO 230 ELSE W=VAL(W$)
750 IF W<32 OR W>143 THEN 740 ELSE W=INT
(W/4)*4
760 FOR I=0 TO 3 :: CALL CHAR(132+I,CHAR
```

continued on page 137

at line 1350 to convert each row of the grid into a binary code, which will, in turn, be converted into a hex code. The hex code will be stored in BYTE$. These hex codes will be stored in PATTERN$. The contents of PATTERN$ will be added to the contents of B$.

Line 650 displays the hex code for that row on the screen. The loop continues until all the rows of the grid have been converted into hex codes, the codes have been stored in P$, and P$ has been printed on the screen. The new character is placed into the string array that stores the characters, and it is printed on the screen. The computer will go back to the program line that displays the menu for this option. You can continue to alter this character, or choose a new character.

Line 670 checks the value of K for 87. If it is an 87, then the W key has been pressed. The value of the T variable is changed to 143. This is the character number of the character that prints an empty square on the screen.

Line 680 checks for the code for the Z. If the Z has been pressed, the T variable is set to 140 so that the character for the filled-in square can be printed on the screen.

Line 690 checks for the R. If the R has not been pressed, the computer will be sent to line 520 so that another key can be pressed.

Line 700 sets the RN variable to one. This variable counts the row numbers as the grid is cleared. The subroutine at line 1270 is used to print the empty grid on the screen. The computer will proceed to line 500 to move the cursor and create a new character on an empty grid. This option is good to use when you are creating a new character that will not resemble the character that it is replacing.

Line 720 begins the routine for displaying four characters the way they would appear if they were used together as sprites. The name of this option is printed at the top of the screen.

Line 730 uses the subroutine at line 1470 to print the grid on the screen. This time the grid will be 16×16 to accommodate four characters.

Line 740 uses the subroutines at line 1510 and 1540 to get the value of one of the characters in the group of four that you want displayed on the screen. The sprite or four-character mode displays the characters in groups of four with each group beginning on an even boundary. This means that the character number that you enter can be displayed in any of the four-character grids on the screen. You cannot move the character into a particular grid. You can change the character in any grid to any design that you would like.

Line 750 checks the value of the character entered to make sure that it is a character that can be changed. If it isn't, the computer will be sent back to line 740 and wait for another number. If it can be changed, the value of the character is divided by four and multiplied by four. This gives us the value of the character on the boundary. This character will be printed in the upper left portion of the grid.

Line 760 changes the characters in locations 132 to 135 to the characters that will be printed on the screen. The characters are taken from CHAR$. If any of these characters have been changed, the new character will be printed on the screen.

Line 770 prints all four characters on the screen. This is the way the characters would be printed on the screen if they were used as a by-four sprite.

Line 780 sets the default values for the color and the magnification of the sprite and prints it on the screen. The characters will be printed in black and the magnification of the sprite will be normal. The value for a normal by-four sprite is three.

Lines 800-870 print all four characters on the grid. You will be able to alter these characters just as you were able to alter the single character.

Line 880 displays the menu. Because this option uses most of the screen space, the menu will cycle every time you press the ENTER key without entering a number. The first option, change color, is on the screen.

Line 890 checks the value of W$ to see if it is a one, or an empty string. If it is neither, the computer will return to the previous line until either a one is entered, or the ENTER key is pressed without the number.

Line 900 displays the second option, change magnify.

```
$(W+I-32))!! NEXT I
770 DISPLAY AT(10,1)SIZE(2)!CHR$(132)&CH
R$(134)!! DISPLAY AT(11,1)SIZE(2)!CHR$(1
33)&CHR$(135)
780 CLR=2 !! M=3 !! W$="COLR" !! R=13 !!
 GOSUB 1510 !! W$="SPRT" !! R=16 !! GOSU
B 1510 !! W$="MAG" !! R=17 !! GOSUB 1510
790 GOSUB 1520 !! GOSUB 1530
800 FOR CH=0 TO 1 !! FOR R=1 TO 8 !! B$=
SEG$(CHAR$(W-32+CH),R*2-1,2)!! GOSUB 130
0
810 FOR COL=1 TO 8 !! BT=VAL(SEG$(BN$,CO
L,1))!! IF BT THEN DISPLAY AT(R+(CH*8)+5
,COL+6)SIZE(1)!CHR$(143)!! GOTO 830
820 DISPLAY AT(R+(CH*8)+5,COL+6)SIZE(1)!
CHR$(140)
830 NEXT COL !! DISPLAY AT(R+(CH*8)+5,24
)SIZE(3)!B$;"-" !! NEXT R !! NEXT CH
840 FOR CH=0 TO 1 !! FOR R=1 TO 8 !! B$=
SEG$(CHAR$(W-30+CH),R*2-1,2)!! GOSUB 130
0
850 FOR COL=1 TO 8 !! BT=VAL(SEG$(BN$,CO
L,1))!! IF BT THEN DISPLAY AT(R+(CH*8)+5
,COL+14)SIZE(1)!CHR$(143)!! GOTO 870
860 DISPLAY AT(R+(CH*8)+5,COL+14)SIZE(1)
!CHR$(140)
870 NEXT COL !! DISPLAY AT(R+(CH*8)+5,27
)SIZE(2)!B$ !! NEXT R !! NEXT CH
880 DISPLAY AT(23,1)!"PRESS [ENTER] ONLY
 TO CYCLE" !! DISPLAY AT(24,1)!"TYPE 1 T
O CHANGE COLOR" !! ACCEPT AT(24,27)BEEP
SIZE(1)!W$
890 IF W$="1" THEN 960 ELSE IF W$="" THE
N 900 ELSE 880
900 DISPLAY AT(24,1)!"TYPE 2 TO CHANGE M
AGNIFY" !! ACCEPT AT(24,27)BEEP SIZE(1)!
W$
910 IF W$="2" THEN 960 ELSE IF W$="" THE
N 920 ELSE 900
920 DISPLAY AT(24,1)!"TYPE 3 TO CHANGE C
ODE" !! ACCEPT AT(24,27)BEEP SIZE(1)!W$
930 IF W$="3" THEN 960 ELSE IF W$="" THE
N 940 ELSE 920
```

continued on page 139

Line 910 checks the W$ variable for the number two or an empty string. Anything else will send the computer back to line 900.

Line 920 places the third option on the screen, change code.

Line 930 compares the contents of W$ with the number 3 and an empty string.

Line 940 is the fourth option, alter characters. Use this option to create new characters on the screen.

Line 950 checks for a four or an empty string. These lines will continue to cycle until one of the numbers is entered.

Line 960 sends the computer to the correct line based on the value of W$. These routines are very similar to the routines used for the single character. The only difference is any or all of the four characters on the screen can be changed.

Line 970 uses the subroutines at lines 1600 and 1520 to change the colors of the characters on the screen.

Line 980 uses the subroutines at lines 1630 and 1530 to change the size of the sprites. In the by-four mode, the three is used for normal sprites and four for enlarged sprites.

Lines 1000-1220 allow you to change the characters on the screen. The commands for this option are the same as the commands for the alter single character mode. We cannot reuse the same routines, however, because the single character mode uses only eight squares in the grid and this routine uses 16. Also, the placement of the grid, the hex values, and the number of values printed on the screen are different for this option. However, the cursor is the same, and the same keys are used to move the cursor and change the pixels on the screen.

Lines 1220-1230 save the character set out to disk. You will give the character set a name. You will use this name to bring the character set back into the computer for future alterations or for use in your own program.

Lines 1250-1260 load a character set from disk into the computer. If you have used this program and saved a character set onto the disk, you can use this option to load the character set back into the computer and make more changes to it. In Chapter 16 we will talk about how these characters can be loaded directly into a different program for immediate use.

Lines 1270-1700 are the subroutines that have been used in the main portion of this program.

Lines 1270-1280 place the grid for the single mode on the screen. The value of each pixel is printed across the top of the screen. By taking the STR$ of the RN variable, we eliminate the leading and trailing spaces that are normally printed with a variable. The RPT$ command places eight characters on the screen. The code for these characters is 140, which makes the empty grid. Finally character 138 is printed on the screen to complete the last box in each row.

Lines 1300-1330 convert the hex value on B$ to an eight-bit binary string. When the computer is sent to this routine, a two-character hex code will be in B$. The computer takes the first character and places it into NY$. It then goes to line 1320 where it gets the ASCII value of the character. This value is used to get the four-bit binary equivalent from HEX$. The computer uses this routine twice, once for each character in B$. When it returns to the main program, BN$ will contain the binary code for B$.

Lines 1350-1380 look at a row in the grid and convert the pixels that are turned on into a binary code. The GCHAR command gets the value of the character at location OLR,COL+2. If the value of this character is 143, then the pixel is darkened or turned on and one is added to the BN$. If it is not 143, a zero is added to the string. After the row is converted into a binary string, the computer divides it into two nibbles, each four bits long, and converts the binary code into hex. When the computer returns to the main program, the hex value of that row will be in the B$.

Lines 1400-1430 count from zero to 15, looking for a match between the contents of NY$ and HEX$ at any of those locations. When both match, the computer is directed to line 1420 where the hex value is placed in HX$. If the value of C is none or less, then that value can be placed in HX$. If it is greater than nine, the value must be converted to

```
940 DISPLAY AT(24,1):"TYPE 4 TO ALTER CH
ARS." :: ACCEPT AT(24,27)BEEP SIZE(1):W$
950 IF W$="4" THEN 960 ELSE IF W$="" THE
N 880 ELSE 940
960 DISPLAY AT(23,1):" " :: ON VAL(W$)GO
TO 970,980,740,1000
970 GOSUB 1600 :: GOSUB 1520 :: GOTO 880
980 GOSUB 1630 :: GOSUB 1530 :: GOTO 880
990 REM ALTER FOUR CHARACTER/SPRITE PATT
ERNS
1000 DISPLAY AT(2,2):"ALTER 4 CHARACTER
PATTERNS"
1010 CR,OLR=6 :: CC,OC=7 :: CUR$=CHR$(14
1)
1020 CALL GCHAR(OLR,OC+2,T)
1030 DISPLAY AT(CR,CC)SIZE(1):CUR$
1040 CALL KEY(0,K,S):: IF S<>0 THEN 1070
1050 IF CUR$=CHR$(141)THEN CUR$=CHR$(139
)ELSE CUR$=CHR$(141)
1060 GOTO 1030
1070 IF K=82 OR K=87 OR K=90 THEN 1160
1080 DISPLAY AT(OLR,OC)SIZE(1):CHR$(T)
1090 IF K=69 THEN OLR=OLR-1 :: IF OLR=5
THEN OLR=21
1100 IF K=88 THEN OLR=OLR+1 :: IF OLR=22
 THEN OLR=6
1110 IF K=68 THEN OC=OC+1 :: IF OC=23 TH
EN OC=7
1120 IF K=83 THEN OC=OC-1 :: IF OC=6 THE
N OC=22
1130 IF K=81 THEN 1200 ELSE IF K<>13 THE
N CR=OLR :: CC=OC :: GOTO 1020
1140 GOSUB 1660 :: W=W-4 :: GOTO 1200
1150 REM SET, CLEAR, OR ERASE
1160 IF K=87 THEN T=143
1170 IF K=90 THEN T=140
1180 IF K<>82 THEN 1030
1190 GOSUB 1470 :: GOTO 1030
1200 DISPLAY AT(2,1):"   DISPLAY FOUR PA
TTERNS" :: GOTO 880
1210 REM SAVE CHARACTER SET
1220 DISPLAY AT(2,6)ERASE ALL:"SAVE CHAR
ACTER SET" :: W$="SAVE" :: N$="" :: GOSU
```

continued on page 141

the correct HEX letter.

Lines 1440-1450 change the character and sprite colors and place the color word on the screen.

Line 1460 changed the size of the sprite.

Lines 1470-1490 place the 16×16 grid on the screen. The values of the pixels are printed across the top of the grid and the empty grid is placed on the screen.

Line 1510 will place a word of any length anywhere on the screen. The loop counts from one to the last letter in the word. The ASCII value of a letter in the word is stored in WC, then this letter is printed on the screen at the location specified by the R and C variables.

Lines 1520-1530 are used to change the color of the character and sprites from the by-four mode.

Line 1540 accepts a code for the character that will be displayed on the screen. The computer will return to the main program with value entered in W$.

Line 1560 gets the name of the character set that you want to save or load.

Line 1580 clears the bottom five rows on the screen.

Lines 1600-1610 get the value of the new color for the character and sprite. If the number entered is less than one or greater than 16, the computer will stay at this line until a correct number is entered. The computer returns to the main program with the color value stored in the CLR variable.

Lines 1630-1640 accept a value to change the size of the sprite. Only the numbers one to four can be entered. Numbers one and two should be used for the single character mode and three and four for the by-four mode. The computer returns to the main program with the magnification value in the M variable.

Lines 1660-1700 change the hex codes of the screen to the new codes for the by-four mode. The character set is also changed to reflect the new characters.

This program should be used from a cold start. That is, the computer should be shut off, then turned back on before using this program. One reason for doing this is the characters that are moved into CHAR$ are the characters from the character set of the TI. If the program that you were using altered any of these characters, the altered characters would show up on the screen instead of the true TI characters.

This program is written to run with the disk system. It can, however, work with the cassette if you change the following lines:

{LINES FOR LOADING AND SAVING CHARACTER SET WITH CASSETTE.}

Line 1220 prints the option at the top of the screen.

Lines 1230-1236 saves the character set to the cassette recorder. Before the values for the new characters can be saved, we need to *pack* the information together so it will be saved quickly and efficiently. We will use CM$ as a temporary storage area for the character codes. Each element of CM$ will contain the codes for 12 characters. Each character contains 16 hex codes. The length of each string in each element of CM$ will be 192 bytes. The two nested FOR . . . NEXT loops will take each byte from the character patterns stored in CHAR$, put them together or concatenate them, and store them in CM$. First the C variable is set to zero. This variable is used as a pointer. It tells the computer which character pattern is being transferred. The first FOR . . . NEXT loop counts from one to nine, the first nine elements of CM$ array where the packed code will be stored. C$ is cleared so that we won't be adding to previously stored information. The next FOR . . . NEXT loop begins with the value of C and continues until N is more than C+11. We will be moving 12 character patterns into each string. The first time this loop is used, the character pattern for the zero element of CHAR$ is stored in C$. The second time this loop is used, N will equal one so the character pattern for the first element of CHAR$ will be added to the character pattern in C$. The loop will continue until the first 12 character patterns (from zero-11) have been placed in C$. This set of patterns is then placed in the element of CM$ that is pointed to by RN. The first 12 character patterns will be placed in the first element of CM$. The C variable has 12 added

```
B 1560 :: IF N$="" THEN 230
1230 OPEN #1:"DSK1."&N$ :: FOR C=0 TO 11
1 :: PRINT #1:CHAR$(C):: NEXT C :: CLOSE
 #1 :: GOTO 230
1240 REM LOAD CHARACTER SET
1250 DISPLAY AT(2,6)ERASE ALL:"LOAD CHAR
ACTER SET" :: W$="LOAD" :: N$="" :: GOSU
B 1560 :: IF N$="" THEN 230
1260 OPEN #1:"DSK1."&N$ :: FOR C=0 TO 11
1 :: INPUT #1:CHAR$(C):: NEXT C :: CLOSE
 #1 :: GOTO 230
1270 DISPLAY AT(6,11)SIZE(8):"84218421"
:: FOR R=8 TO 15 :: DISPLAY AT(R,5)SIZE(
5):"ROW "&STR$(RN):: RN=RN+1
1280 DISPLAY AT(R,11)SIZE(9):RPT$(CHR$(1
40),8)&CHR$(138):: NEXT R :: DISPLAY AT(
R,11)SIZE(8):RPT$(CHR$(137),8):: RETURN
1290 REM SUBROUTINE TO CONVERT 2-DIGIT H
EX CODE TO 8-BIT BINARY
1300 BN$="" :: NY$=SEG$(B$,1,1):: GOSUB
1320 :: NY$=SEG$(B$,2,1)
1310 REM SUBROUTINE TO CONVERT 1-DIGIT H
EX CODE TO BINARY
1320 NY=ASC(NY$):: IF NY>57 THEN NY=NY-5
5 ELSE NY=NY-48
1330 BN$=BN$&HEX$(NY):: RETURN
1340 REM CONVERT 1 ROW OF BLOCKS TO BINA
RY CODE
1350 BN$="" :: FOR COL=SC TO EC :: CALL
GCHAR(OLR,COL+2,T):: IF T=143 THEN BN$=B
N$&"1" ELSE BN$=BN$&"0"
1360 NEXT COL
1370 REM CONVERT 8-BIT BINARY CODE TO 2-
DIGIT HEX CODE
1380 B$="" :: NY$=SEG$(BN$,1,4):: GOSUB
1400 :: B$=B$&HX$ :: NY$=SEG$(BN$,5,4)::
 GOSUB 1400 :: B$=B$&HX$ :: RETURN
1390 REM CONVERT A BINARY NYBBLE TO HEX
CODE
1400 FOR C=0 TO 15 :: IF NY$=HEX$(C)THEN
 1420
1410 NEXT C
1420 IF C>9 THEN HX$=CHR$(C+55)ELSE HX$=
```

```
CHR$(C+48)
1430 RETURN
1440 CALL COLOR(13,CLR,1):: DISPLAY AT(8
,26):CHR$(135):: CALL SPRITE(#1,135,CLR,
104,217):: L=LEN(COLR$(CLR,1))
1450 DISPLAY AT(19,1):" " :: DISPLAY AT(
19,28-L):COLR$(CLR,1):: RETURN
1460 DISPLAY AT(12,27):STR$(M):: CALL MA
GNIFY(M):: RETURN
1470 RN=1 :: DISPLAY AT(4,7)SIZE(16):RPT
$("84218421",2):: FOR R=6 TO 21 :: IF R=
6 THEN DISPLAY AT(R,1)SIZE(4):"ROW:"
1480 DISPLAY AT(R,5)SIZE(1):STR$(RN):: R
N=RN+1 :: IF RN>8 THEN RN=1
1490 DISPLAY AT(R,7)SIZE(17):RPT$(CHR$(1
40),16)&CHR$(138):: NEXT R :: DISPLAY AT
(R,7)SIZE(16):RPT$(CHR$(137),16):: RETUR
N
1500 REM ROUTINE TO PLACE WORDS ANYWHERE
 ON THE SCREEN
1510 L=LEN(W$):: FOR C=1 TO L :: WC=ASC(
SEG$(W$,C,1)):: CALL HCHAR(R,C+1,WC,1)::
 NEXT C :: RETURN
1520 CALL COLOR(13,CLR,1):: CALL SPRITE(
#1,132,CLR,144,17):: W$=COLR$(CLR,2):: R
=14 :: GOTO 1510
1530 DISPLAY AT(17,3)SIZE(1):STR$(M):: C
ALL MAGNIFY(M):: RETURN
1540 DISPLAY AT(24,1):"ENTER CHAR. CODE(
32-143)?" :: ACCEPT AT(24,26)BEEP VALIDA
TE(DIGIT)SIZE(3):W$ :: RETURN
1550 REM GET FILE NAME
1560 DISPLAY AT(12,1):"ENTER NAME OF CHA
RACTER SET TO ";W$;" :" :: ACCEPT AT(13,
11)BEEP SIZE(10):N$ :: RETURN
1570 REM CLEAR BOTTOM 5 LINES
1580 FOR R=20 TO 24 :: DISPLAY AT(R,1):"
 " :: NEXT R :: RETURN
1590 REM GET NEW COLOR
1600 DISPLAY AT(24,1):"ENTER COLOR CODE(
1-16) ?" :: ACCEPT AT(24,26)BEEP VALIDAT
E(DIGIT)SIZE(2):CLR :: IF CLR<1 OR CLR>1
6 THEN 1600
```

continued on page 143

```
1610 DISPLAY AT(24,1):" " :: RETURN
1620 REM GET NEW MAGNIFICATION
1630 DISPLAY AT(24,1):"MAGNIFICATION FAC
TOR(1-4)?" :: ACCEPT AT(24,28)BEEP VALID
ATE(DIGIT)SIZE(1):M :: IF M<1 OR M>4 THE
N 1630
1640 DISPLAY AT(24,1):" " :: RETURN
1650 REM CONVERT NEW CHARACTER TO HEX-CO
DE & INSERT IN CHAR$(W)
1660 A,A2=0 :: SR=6 :: ER=13 :: A1=1 ::
GOSUB 1680 :: SR=14 :: ER=21 :: GOSUB 16
80
1670 A2=3 :: SR=6 :: ER=13 :: A1=9 :: GO
SUB 1680 :: SR=14 :: ER=21
1680 P$="" :: FOR OLR=SR TO ER :: SC=6+1
*A1 :: EC=13+1*A1 :: GOSUB 1350 :: GOSUB
 1400 :: P$=P$&B$
1690 DISPLAY AT(OLR,24+A2)SIZE(2):B$ ::
NEXT OLR :: CHAR$(W-32)=P$ :: CALL CHAR(
132+A,P$)
1700 W=W+1 :: A=A+1 :: RETURN


1220 DISPLAY AT(23,6)ERASE ALL:"SAVE CHA
RACTER SET"
1230 C=0 :: FOR RN=1 TO 9 :: C$="" :: FO
R N=C TO C+11 :: C$=C$&CHAR$(N):: NEXT N
1232 CM$(RN)=C$ :: C=C+12 :: NEXT RN ::
C$="" :: FOR N=C TO C+3 :: C$=C$&CHAR$(N
)
1234 NEXT N :: CM$(10)=C$
1236 OPEN #1:"CS1",OUTPUT,FIXED 192 :: F
OR RN=1 TO 10 :: PRINT #1:CM$(RN):: NEXT
 RN :: CLOSE #1 :: GOTO 230
1250 DISPLAY AT(23,6)ERASE ALL:"LOAD CHA
RACTER SET"
1252 OPEN #1:"CS1",INPUT ,FIXED 192 :: F
OR C=1 TO 10 :: LINPUT #1:CM$(C):: NEXT
C :: CLOSE #1
1254 C=0 :: FOR RN=1 TO 9 :: A=1 :: FOR
N=C TO C+11 :: CHAR$(N)=SEG$(CM$(RN),A,1
6)
1256 A=A+16 :: NEXT N :: C=C+12 :: NEXT
RN :: A=1 :: FOR N=C TO C+3
1260 CHAR$(N)=SEG$(CM$(RN),A,16):: A=A+1
6 :: NEXT N :: GOTO 230
```

Table 14-1
Table of Commands for Listing 14-3.

(The ALPHA LOCK KEY must be down)

KEY	FUNCTION	KEY	FUNCTION
E (up arrow)	Cursor up	Z	Whiten square
X (down arrow)	Cursor down	R	Erase all blocks
D (right arrow)	Cursor right	Q	Quits alter mode—no changes
S (left arrow)	Cursor left	{ENTER}	Make changes on screen—leave alter mode
W	Blacken square		

to it, so now it can point to the second set of 12 characters in CHAR$. Because RN will be two, the second set of twelve character patterns (12 to 23) will be stored in the second element of CM$. Both FOR . . . NEXT loops continue until nine sets of 12 character patterns each have been packed together and moved into CM$. The next FOR . . . NEXT loop packs the last four character patterns together and places them in the tenth element of CM$ array. Now the entire character set is ready to be saved to cassette. The file is opened to output to the first cassette. Each record will have a fixed length of 192 bytes. The FOR . . . NEXT loop will send out the ten elements of CM$.

Lines 1250-1260 open the file for the cassette to input or read the character set from the cassette at the fixed length of 192 bytes for each record. The FOR . . . NEXT loop uses the LINPUT or line input command to get the records from the cassette. The LINPUT command is used due to the length of the records. The records are stored in the string array CM$. Follow the instructions on your screen to turn the cassette recorder on and off. Once these records are in the computer, the next FOR . . . NEXT loop breaks down each of these 10 records into the 12 characters that are stored on them. Each character is made of a 16 hex digit code. The SEG$ command removes 16 codes or one character pattern at a time and moves it into CHAR$. There are 12 character patterns stored in the first nine records. The tenth record contains last four characters.

When the computer is saving the characters to the cassette it does not display the message RECORDING on the screen. You can hear your characters being saved by turning up the volume on your monitor or television. You should hear a sound like a burst of noise separated by a few seconds of tone.

Chapter 15
Finding and Trapping Errors

The most necessary steps in programming are testing and debugging your program. When you write a program, you are familiar with its functions, what answers or inputs are expected, and how the program is supposed to work. The best test for a program is to let someone who is unfamiliar with the program sit at the computer and try it. It's amazing how many errors can appear when someone else is using your program. Of course, there are some errors the program cannot check for. If you enter 50 instead of 5, you would not expect the program to ask, "ARE YOU SURE?" after every question. On the other hand, there are ways to check for errors before or after they happen, have the program recover from the error and avoid having the user experience an error message on the screen.

ERROR-TRAPPING TECHNIQUES

ON ERROR

The ON ERROR command is one of the commands that handles an error that may occur in a program. When a program error occurs, the error is printed on the screen and the program stops. If it is possible for the user to enter any type of information that could result in an error, the program should be aware of the possible error and check for it. It is also possible for the error to be generated by the program itself. Again, the program can be set up to watch for an error and handle it.

The ON ERROR statement is similar to a GOSUB in that it also needs a RETURN statement after the error is handled. ON ERROR must be followed by a line number. This line number is the program line that the computer will go to if an error occurs.

Try the program in Listing 15-1 without the ON ERROR instruction in line 140. After the words have been printed on the screen, and the computer has no more data, an error message is printed on the screen, and the program stops. Now put the ON ERROR line back in. The program will continue until the FCTN and 4 keys are pressed. When the computer runs out of data, it is in error. This time,

Listing 15-1

```
100 REM LISTING 15-1
110 REM ON ERROR
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DISPLAY AT(2,2)ERASE ALL:"I'M GOING
TO READ MY OWN":"DATA."
140 ON ERROR 180
150 READ A$ :: DISPLAY AT(7,7):"I READ,
";A$
160 FOR TIME=1 TO 500 :: NEXT TIME
170 GOTO 150
180 RESTORE :: RETURN 140
190 DATA FATHER,MOTHER,BROTHER,SISTER,AU
NT,UNCLE,COUSIN
```

however, line 135 tells the computer that when an error occurs, go to line 180. The computer does this and finds a RESTORE statement. Now it can continue reading the data and printing the words on the screen. The RETURN tells the computer to go back to line 140 before continuing, which resets the error handling.

Remove the line number after the RETURN and run the program again. This time the RETURN will send the computer back to the line where the error occurred. The computer will read the words in the DATA line and print them on the screen. The error occurs again and the program stops. Once an ON ERROR has been handled, it is cleared from the computer. The error handling command must be executed again when the next error occurs.

ON BREAK

You can protect your programs from being interrupted with the ON BREAK command, by using the ON BREAK NEXT command at the beginning of the program. One word of warning here—with the ON BREAK NEXT command, only a break point set within the program itself will stop the program. If the program loops, and there is no natural end to the program, the only way to stop it is to use the quit function.

When you do not specify ON BREAK NEXT, the default is ON BREAK STOP. When the FCTN and 4 keys are pressed, the program will stop. The ON BREAK STOP statement can be used within the program to restore the default.

You can also tell the computer to break or stop at a particular line number. Use BREAK with a line number at the beginning of the program and the computer will stop at that line. If you use BREAK without a line number, the computer will stop immediately upon reading the instruction.

```
150  BREAK
```

The computer will stop when it comes to line 150 no matter what. However, if you use BREAK with the line number before the ON BREAK NEXT command, the computer will not stop at that line number. It will not stop at 150 as directed to in line 110 of the short listing below:

```
110  BREAK 150
120  ON BREAK NEXT
130  PRINT "I WILL"
140  PRINT "KEEP"
150  PRINT "GO....."
160  PRINT "....ING"
170  STOP
```

However, if you remove line 110 and add:

```
155  BREAK
```

the computer will stop at line 155. The following program in Listing 15-2 demonstrates the ON BREAK command.

ON WARNING

Some errors do not result in an error message, but rather a warning message printed near the bottom of the screen. When a warning occurs, the computer allows the user to try again. Go back through some of the programs in this book. You will notice that in most of the program statement where the ACCEPT AT command was used, the program specified whether the entry should be numbers, and if they were which numbers should be accepted. You may have accidentally pressed the ENTER key without entering a number. The VALIDATE option can only check for a valid number. An entry with no number gets a warning message and the opportunity to try again.

There are two different examples of eliminating the warning in the program in Listing 15-3. In the first few lines of the program, you are asked to enter the number for the month that you were born in. The VALIDATE option makes sure that only numbers can be entered. If you press the ENTER key without entering a number, a warning appears on the screen. Now add the ON WARNING line to the program. You can press the enter key all you want. The program will not advance and you will not get an error message.

The second part of the program does display the warning message of the screen because this is the default for an error that gives warning messages. In this routine, you are asked to enter a number under 295. The computer will find the value of e (2.718281828459) raised to the value that you enter. Any number larger than 294 will cause an error. Enter numbers that are less than and greater than 295.

The third way to use the ON WARNING is with the STOP command. When the computer has a value error, instead of displaying the error and waiting for another entry, the program stops after the warning message is displayed. If the ON WARNING STOP or the ON WARNING NEXT is used in a program, it will stay in effect until an ON

Listing 15-2

```
100 REM LISTING 15-2
110 REM ON BREAK
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: PRINT "RUN COMMAND ENT
ERED"
140 GOSUB 370
150 PRINT "DEFAULT OF ON BREAK STOP IN E
FFECT"
160 GOSUB 370 :: BREAK 340 :: PRINT "BRE
AK 320 COMMAND ISSUED" :: GOSUB 370
170 ON BREAK NEXT :: PRINT "ON BREAK NEX
T COMMAND ISSUED"
180 GOSUB 370
190 PRINT "ANY BREAKPOINTS ?        ";::
GOSUB 380 :: PRINT "NO !"
200 GOSUB 370 :: PRINT "CLEAR PRESSED ?"
210 FOR COUNT=1 TO 75 :: CALL KEY(0,KEY,
STATUS)
```

```
220 IF STATUS<>0 OR STATUS<>-1 THEN IF C
OUNT>49 THEN DISPLAY AT(23,17):"TRY IT"
:: GOSUB 380
230 IF KEY=2 THEN DISPLAY AT(23,17):"YES
, HA HA !" :: GOSUB 380 :: GOSUB 380 ::
COUNT=75 ELSE DISPLAY AT(23,17):"NO !"
240 NEXT COUNT :: GOSUB 370
250 PRINT "EXECUTE NEXT COMMAND" :: GOSU
B 370
260 PRINT "PROGRAMMED BREAKPOINT HIT":"L
IST LINE # IN MESSAGE AND":"NOTE REMARK"
 :: BREAK ! TYPE 'CON' TO CONTINUE
270 GOSUB 370 :: PRINT "EXECUTE NEXT COM
MAND" :: GOSUB 370
280 PRINT "CLEAR PRESSED ?"
290 FOR COUNT=1 TO 75 :: CALL KEY(0,KEY,
STATUS)
300 IF STATUS<>0 OR STATUS<>-1 THEN IF C
OUNT>49 THEN DISPLAY AT(23,17):"TRY IT"
:: GOSUB 380
310 IF KEY=2 THEN DISPLAY AT(23,17):"YES
, HA HA !" :: GOSUB 380 :: GOSUB 380 ::
COUNT=75 ELSE DISPLAY AT(23,17):"NO !"
320 NEXT COUNT :: GOSUB 370
330 ON BREAK STOP :: PRINT "ON BREAK STO
P COMMAND ISSUED" :: GOSUB 370 :: PRINT
"TYPE 'CON' AFTER BREAKPOINT"
340 GOSUB 370 :: PRINT "EXECUTE NEXT COM
MAND"
350 GOSUB 370 :: PRINT "PRESS CLEAR NOW
TO END" :: GOTO 350
360 END
370 FOR COUNT=1 TO 5 :: PRINT TAB(14);"
" :: NEXT COUNT
380 FOR DELAY=1 TO 500 :: NEXT DELAY ::
RETURN
```

WARNING PRINT is issued. Be sure to clear any error trapping that you have set up before running another program.

RETURN

The RETURN statement is used with ON ERROR. The ON ERROR is similar to a GOSUB. After it executes the lines that it has been sent to, it needs a RETURN to send it back. There is one important difference between the RETURN used here and the GOSUB's RETURN. With the ON ERROR's RETURN, you have three different op-

Listing 15-3

```
100 REM LISTING 15-3
110 REM ON WARNING
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DISPLAY AT(2,1)ERASE ALL:"THIS DEMON
STRATES                  'ON WARNING'"
140 ON WARNING NEXT
150 DISPLAY AT(5,3):"ENTER THE NUMBER OF
 THE":"MONTH THAT YOU WERE BORN"
160 ACCEPT AT(6,26)VALIDATE(DIGIT):MONTH
 :: IF MONTH<1 OR MONTH>12 THEN 160
170 DISPLAY AT(10,5):"YOU DID IT RIGHT!"
180 FOR TIME=1 TO 700 :: NEXT TIME
190 IF EXAMPLE>=5 THEN DISPLAY AT(2,3)ER
ASE ALL:"ON WARNING STOP" :: ON WARNING
STOP :: GOTO 220
200 DISPLAY AT(2,3)ERASE ALL:"ON WARNING
 WITH WARNING"
210 ON WARNING PRINT
220 DISPLAY AT(5,3):"ENTER A NUMBER LESS
 THAN 295"
230 ACCEPT AT(9,8)SIZE(11)VALIDATE(NUMER
IC):N
240 DISPLAY AT(11,10):EXP(N)
250 EXAMPLE=EXAMPLE+1 :: GOTO 180
```

tions to choose from. The RETURN can be used alone. In this case, the computer goes back to the line that was in error. If there is a line number after the RETURN, the computer will go to that line number to continue the program. The word NEXT can also be placed after RETURN. In this case, the computer would go to the program line *immediately following* the one in error. You will have to decide which method works best for your needs.

```
100  RETURN ! TRY THE LINE AGAIN
110  RETURN 30 ! GO TO LINE 30
110  RETURN NEXT ! USE THE LINE AFTER
     THE ONE IN ERROR
```

TESTING FOR ERRORS

As you write your program, you should test every routine and subroutine as it is added to the program. Since every possible situation should be taken into account, and is not always possible to do so, try to test for the extreme situations, such as the largest value you expect, then a larger value; the smallest value that should be entered, then an even smaller value. Decimals, negative numbers, and letters should be trapped, validated, or checked in the program. Check that the FOR . . . NEXT loops exit when and where they should. If there is another exit from the loop, does it branch to the correct routine? Does the GOSUB command return to the correct line, and if a program goes to a subroutine because of an IF . . . THEN statement, is the program correctly branched around the unnecessary lines?

If you are testing a routine that is not working correctly, you should first try break points at the line you think is causing the error. Also set a break point before you enter the routine.

A break point is set by placing the command BREAK in a line, or by telling the computer at the beginning of the program BREAK (line number). When the computer comes to that line number, the program will stop. Check any variables for accuracy before the program enters the routine by printing them to the screen with a direct command. Then type CONT. When the program stops again, check the variables once more. If the variables were correct when the computer entered the routine and are now incorrect, the error is occurring somewhere between the two break points. Set a new break point between the two in the program and try it again. Keep dividing the area between the correct line and the incorrect line until you can pinpoint the error. Of course, if at the second break point the variable(s) were correct, the error occurs after this line. Move the break point to the end of the routine and try again. After you correct the error, remove all the breaks in the program. Then type UNBREAK in the direct mode. If you had a break point set at a particular line in the program and you did not reach that line while you were testing the program, the computer would remember that break point, and break at that line even after the command had been removed from the program. UNBREAK erases or clears all break points that have been set.

Another way to find out why a certain routine is being used when you think it shouldn't be, or why the computer is coming up with strange values is to use the TRACE command. The TRACE command can be used as a direct command.

Type TRACE {ENTER}
then type RUN {ENTER}

The program in the computer will be executed, but all the line numbers that the computer is using will be displayed on the screen while the computer is using them. This is very useful for checking which program lines are being used and which ones aren't. Maybe the GOTO is sending the computer to the wrong line. You will find out by following the line numbers printed on the screen.

With a very long program, you may know that most of the program is working correctly, but one routine seems to be wrong. Place the TRACE command in the program just before the computer enters that routine.

```
110  TRACE
```

Now the program will run without displaying any line numbers until this line is reached. Once this line is executed, the computer will print the line numbers on the screen as it executes that line.

The computer will continue to use the TRACE command until you tell it to stop. In the direct mode, you can type UNTRACE {ENTER}. Under program control, you can use the UNTRACE command with a line number.

```
300  UNTRACE
```

Sometimes the program is operating correctly, but it is not running smoothly. It is taking too long to arrive at the answer, the screen does not look clean, the messages are garbled. These are weak points of the program. If it appears the program is running too slow, try to tighten the code or instructions by placing more than one statement on a line. (Watch out for IF . . . THEN and GOSUB statements—when tightening code you can have problems with them.)

CALL ERR

You can also find out more about the error that has occurred in the program—which line number it occurred at, the type of error, and the severity of the error. The error code is the number of the particular error. For instance, LINE NOT FOUND is an error code 60. If the type of error is a negative number, then the error occurred within the program. The severity code is always nine, and the line number is the line at which the error occurred. This may not be the line that caused the error, but only the line where the error was detected. The format for the CALL ERR is as follows:

```
100  CALL  ERR(CODE,TYPE,SEVERITY,
     LINE)
```

If you are not interested in the line number,

then you do not have to use the last two variables—SEVERITY and LINE. The first two variables will contain the error code and the type of error.

Playing Computer

Sometimes the best way to find an error that does not readily appear when you use the usual methods is with a pencil and paper. Make a list of the variable being used. Write down the line number you are starting with and the value of the variables at that time. As you work each line of the program, change the variables the way the computer would. Calculate the equations and check the lines the program would direct the computer to. When you go to a subroutine, mark the line on the paper, work the subroutine, and return to that line. Many errors are made by reusing a variable in a subroutine that you are using in the main program. The program returns to the main part of the program with a different value and causes an error later in the program. Other times, you find the program has been directed to another line and never returns to the original line at all! By working the program as the computer would, it is easy to spot such mistakes. This method can also alert you to routines used within the program that could be made into subroutines.

Chapter 16

Sights and Sounds

One of the most exciting features of the TI-99/4A computer is its graphics and music capabilities. As you have seen with many of the programs in the book, you are not limited by the characters set within your TI-99/4A. You can change the characters to any form or design you want or need for your program. You are limited only by your imagination. Your TI-99/4A can also be programmed to produce music or sound effects. Once you have added music to your program, you will not want to use the *silent* version again!

USING GRAPHICS COMMANDS

CALL CHAR

The CALL CHAR command has been used in several programs in this book. This is the command that changes a character in the character set into a new character. Each character in the character set is made up of an 8×8 grid or set of pixels. (See Fig. 16-1). The character is arranged so that one row of pixels is one byte, and the character is eight bytes high. In order to change one of these characters, we assign new values to each byte that makes up the character. When you used the Character Pattern program in Chapter 14, you were able to see the byte value of each row on the screen. These eight values are passed to the character set in the computer. Once the values have been changed, that character number will become your new character. Any character from 32 through 143 can be changed. The character code is the first number in the parentheses.

CALL CHAR(34,{pattern})

In the above example, we would change the code of the quotation mark. The pattern is the hex code that makes up the new character. The pattern cannot exceed eight bytes—16 numbers. Trailing zeroes such as those in "FF00," can be omitted. If you enter less than 16 numbers, the computer will use zeroes for the remaining bytes. Try this:

```
10 CALL CHAR(36,FFFFC3A59999A5C3FFFF)
```

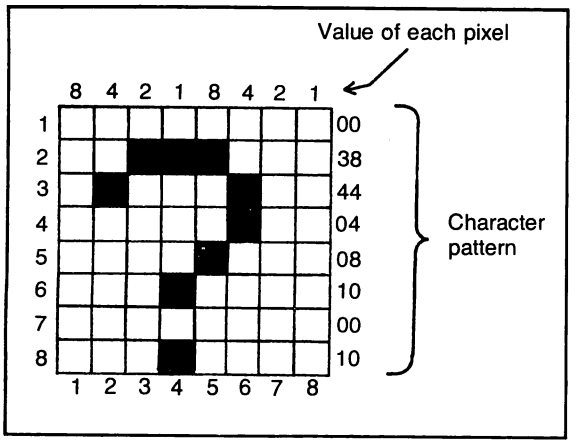


Fig. 16-1 An 8×8 grid for character.

```
20 PRINT "$"
30 GOTO 30
```

When the computer prints the dollar sign ($), an "X" inside a box should be printed instead.

CALL CHARPAT

This command is the opposite of the CALL CHAR command. This command tells you what the pattern of a particular character is. Its format is: CALL CHARPAT(36,B$). The first number in the parentheses is the code of the character that you want. The string variable will contain the character pattern for that code. Enter this in the direct mode.

```
CALL CHARPAT(36,B$)
```

Now type PRINT B$. B$ will contain the character pattern for the dollar sign. If you have changed the character, the code should be the code that you just entered to make the "X" in the box. If you didn't change the character, the character pattern that you see is for the dollar sign.

CALL CHARSET

This command restores the original character set. When you RUN a program, the character set is not restored. Any characters that you may have changed in one program will remain changed until this command is executed (or you quit and restart). If you do not want to reuse your new character set, be sure to have this command in it. Of course, once the characters have been restored to normal, they can be changed again.

CALL HCHAR

The HCHAR makes printing the same character several times in a row very easy. All you need to do is tell the computer the row that it should be printed in, the column to start in, the character code, and the number of times that character should be printed. The row number can be any number from one to 24. The column can be a number from one to 32. The character code can be any value from zero to 32767. Although there are no actual characters past 255, the computer will convert the code to a value between 0 and 255. For example, 289 would be converted to 33 and the exclamation point would be printed. This command along with the next command can be used to make a border around a menu or with a FOR . . . NEXT loop to make a design.

CALL VCHAR

This command is very similar to the HCHAR command except that it prints the character in a column on the screen. The format is the same as that of CALL HCHAR–CALL VCHAR(row number, column number, character code, number of characters). The program in Listing 16-1 (flowcharted in Fig. 16-2) illustrates these two commands.

Listing 16-1

Line 130 will trap the computer for warning errors. If an error occurs, the computer will continue with the program.

Line 150 prints the format for the HCHAR command on the screen.

Line 160 prints the line that will be filled with the proper numbers.

Line 170 asks for a row number. The computer will accept any number between one and 24 inclusive.

Line 180 places the number entered in the command and asks for the column number.

Line 190 makes sure that the number entered is

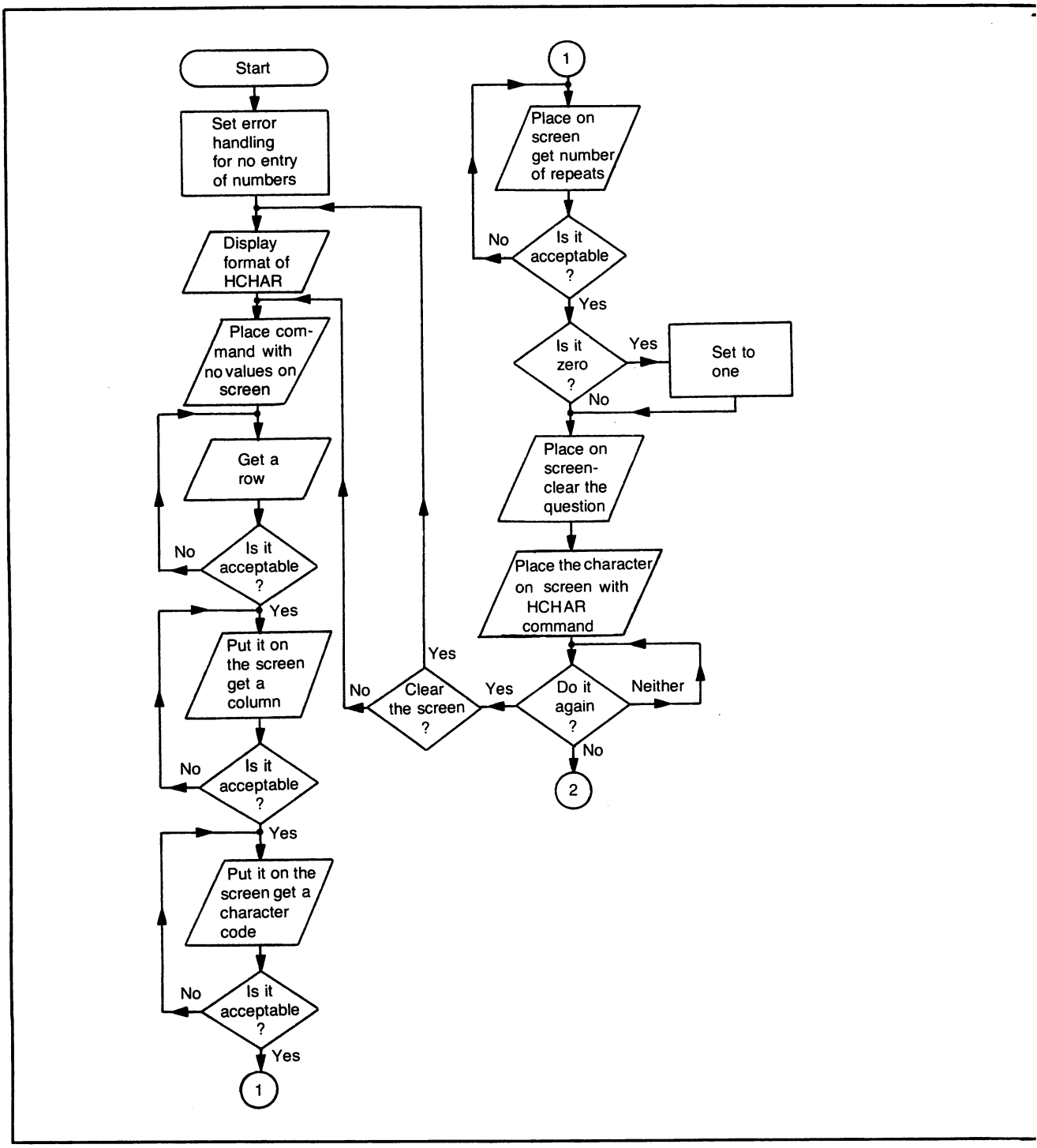
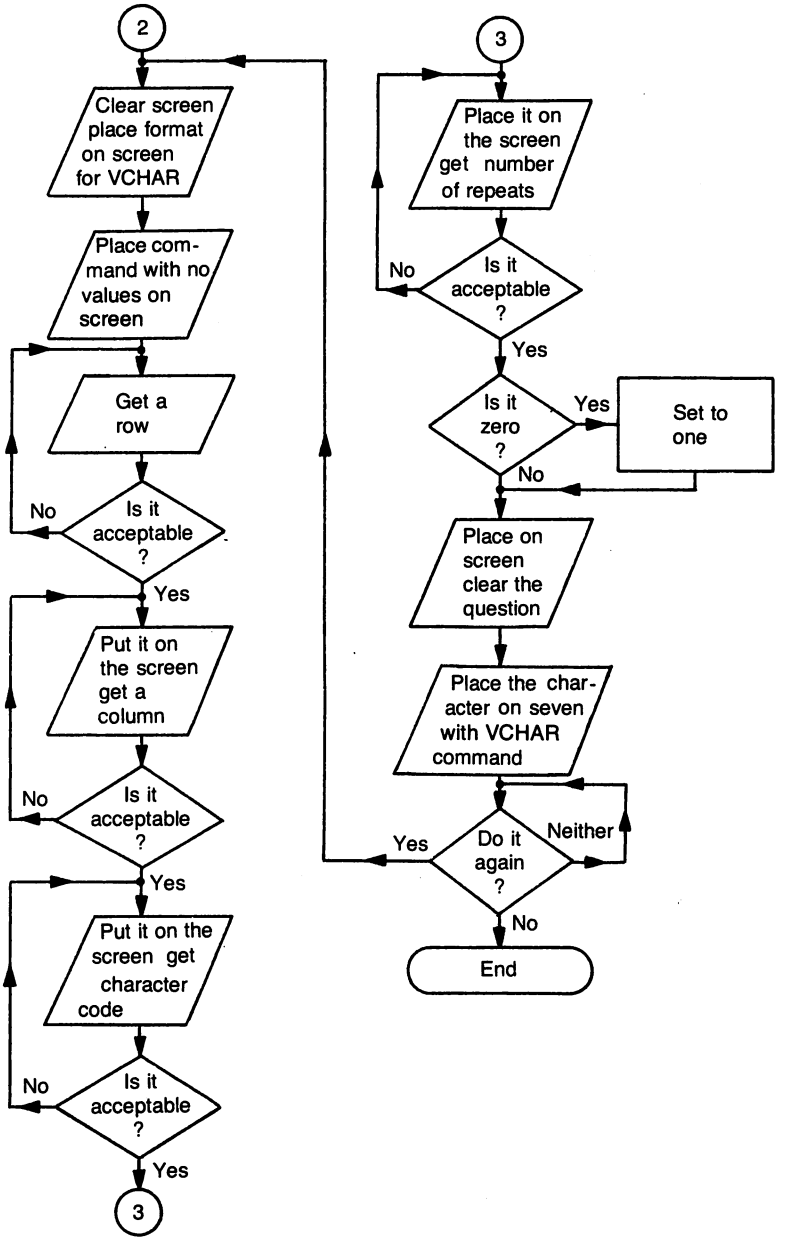


Fig. 16-2. Flowchart for Listing 16-1 HCHAR/VCHAR Example.


2
Clear screen place format on screen for VCHAR
Place command with no values on screen
Get a row
No
Is it acceptable ?
Yes
Put it on the screen get a column
No
Is it acceptable ?
Yes
Put it on the screen get character code
No
Is it acceptable ?
Yes
3
3
Place it on the screen get number of repeats
No
Is it acceptable ?
Yes
Is it zero ?
Yes
Set to one
No
Place on screen clear the question
Place the character on seven with VCHAR command
Yes
Do it again ?
Neither
No
End

Listing 16-1

```
100 REM LISTING 16-1
110 REM HCHAR/VCHAR EXAMPLE
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 ON WARNING NEXT
140 REM HCHAR
150 DISPLAY AT(2,11)ERASE ALL:"HCHAR": :
"FORMAT:": :"CALL HCHAR(ROW,COL,CHAR COD
E":" [,REPETITION])"
160 DISPLAY AT(10,1):"CALL HCHAR(  ,  ,
  [,   ])"
170 DISPLAY AT(12,1):"GIVE ME A ROW #(1-
24)?" :: ACCEPT AT(12,24)BEEP VALIDATE(D
IGIT)SIZE(2):ROW :: IF ROW<1 OR ROW>24 T
HEN 170
180 DISPLAY AT(10,12)SIZE(2):STR$(ROW)::
 DISPLAY AT(12,11):"COLUMN #(1-32)?" ::
ACCEPT AT(12,27)BEEP VALIDATE(DIGIT)SIZE
(2):COLUMN
190 IF COLUMN<1 OR COLUMN>32 THEN 180
200 DISPLAY AT(10,15)SIZE(2):STR$(COLUMN
):: DISPLAY AT(12,11):"CHAR CODE(0-255)"
:" ?" :: ACCEPT AT(13,4)BEEP VALIDATE(DI
GIT)SIZE(3):CODE
210 IF CODE<0 OR CODE>255 THEN 200
220 DISPLAY AT(10,18)SIZE(3):STR$(CODE):
: DISPLAY AT(12,11):"REPEAT #(0-768)":"
?" :: ACCEPT AT(13,4)BEEP VALIDATE(DIGIT
)SIZE(3):REPETITION
230 IF REPETITION<0 OR REPETITION>768 TH
EN 220 ELSE IF REPETITION=0 THEN REPETIT
ION=1
240 DISPLAY AT(12,1):RPT$(" ",38)
250 DISPLAY AT(10,23)SIZE(3):STR$(REPETI
TION)
260 CALL HCHAR(ROW,COLUMN,CODE,REPETITIO
N)
270 DISPLAY AT(24+(ROW=24),1):"DO ANOTHE
R (Y/N) ?" :: ACCEPT AT(24+(ROW=24),20)B
EEP VALIDATE("YN")SIZE(1):A$ :: IF A$=""
 THEN 270
280 IF A$="N" THEN 310 ELSE DISPLAY AT(2
4+(ROW=24),1):" "
290 IF ROW<11 OR REPETITION>(25-ROW)*32-
```

```
COLUMN+1 THEN 150 ELSE GOTO 160
300 REM VCHAR
310 DISPLAY AT(2,11)ERASE ALL:"VCHAR": :
"FORMAT:": :"CALL VCHAR(ROW,COL,CHAR COD
E":" [,REPETITION])"
320 DISPLAY AT(10,1):"CALL VCHAR(   ,   ,
  [,    ])"
330 DISPLAY AT(12,1):"GIVE ME A ROW #(1-
24)?" :: ACCEPT AT(12,24)BEEP VALIDATE(D
IGIT)SIZE(2):ROW :: IF ROW<1 OR ROW>24 T
HEN 330
340 DISPLAY AT(10,12)SIZE(2):STR$(ROW)::
 DISPLAY AT(12,11):"COLUMN #(1-32)?" ::
ACCEPT AT(12,27)BEEP VALIDATE(DIGIT)SIZE
(2):COLUMN
350 IF COLUMN<1 OR COLUMN>32 THEN 340
360 DISPLAY AT(10,15)SIZE(2):STR$(COLUMN
)):: DISPLAY AT(12,11):"CHAR CODE(0-255)"
:" ?" :: ACCEPT AT(13,4)BEEP VALIDATE(DI
GIT)SIZE(3):CODE
370 IF CODE<0 OR CODE>255 THEN 360
380 DISPLAY AT(10,18)SIZE(3):STR$(CODE):
: DISPLAY AT(12,11):"REPEAT #(0-768)":"
?" :: ACCEPT AT(13,4)BEEP VALIDATE(DIGIT
)SIZE(3):REPETITION
390 IF REPETITION<0 OR REPETITION>768 TH
EN 380 ELSE IF REPETITION=0 THEN REPETIT
ION=1
400 DISPLAY AT(12,1):RPT$(" ",38)
410 DISPLAY AT(10,23)SIZE(3):STR$(REPETI
TION)
420 CALL VCHAR(ROW,COLUMN,CODE,REPETITIO
N)
430 DISPLAY AT(24,1):"DO ANOTHER (Y/N) ?
" :: ACCEPT AT(24,20)BEEP VALIDATE("YN")
SIZE(1):A$ :: IF A$="" THEN 430
440 IF A$="N" THEN 450 ELSE DISPLAY AT(2
4,1):" " :: GOTO 310
450 END
```

between one and 32 inclusive.

Line 200 places the column number in the command and asks for the character code. Although the computer will accept any code from zero to 32767, this program will only accept character codes between zero and 255.

Line 210 checks the value of the CODE variable. If it is not within the limits, the computer will be

directed back to line 200.

Line 220 places the code in the command and asks for the number of times you would like this character to be printed. This program will accept any number from zero to 768.

Line 230 checks the value of the REPETITION variable. If it is not a valid number, the computer will be directed back to line 220. If it is a zero, the computer will substitute a one for the number.

Line 240 erases the last question from the screen. The RPT$ tells the computer to print a series of 38 spaces beginning with the first column in the 12th row.

Line 250 places the value of REPETITION in the command on the screen.

Line 260 uses the HCHAR command to display the character that you entered at the row and column specified for as many times as you indicated.

Line 270 asks if you would like to try another set of values. This question will be printed on the 24th row unless that row was the one specified to begin the characters printed. The program does not use an IF . . . THEN command to test the row number. Instead it uses a logic command. Look at the DISPLAY AT command. After the plus sign, the ROW=24 is in parenthesis. The computer tests the value of the ROW variable. If it is 24, a −1 will be added to the 24. The computer uses the value −1 if the expression within the parentheses is true. If the value of ROW is any other value, a zero will be added to the 24. The computer uses a value of zero when the expression within the parentheses is false. The program will only accept an "N" or a "Y" for the answer. If the ENTER key is pressed and no letter is entered, the computer will remain on this line until the "N" or "Y" is entered. If a "N" was entered, the computer will go on to line 310 for the VCHAR command. Otherwise, it removes the question from the screen.

Line 290 checks the value of the ROW. If the value of ROW is less than 11 or the number of times the character was printed caused it to wrap around to the top of the screen, the message on the screen was written over. The computer will be directed to line 150 to erase the screen and get a new row, column, character code and number of characters. If the characters printed on the screen were not printed over the message, they will be left on the screen when the computer asks for the new values.

Line 310 begins the part of the program that demonstrates the VCHAR command. The format for this command is printed on the screen.

Line 320 places the command on the screen. You will be asked to fill in the blanks.

Line 330 asks for a row number. Again, the row must be between one and 24 inclusive. If it is not, the computer will remain at this line until a correct number is entered.

Line 340 places the row number in the command on the screen. It then asks for the column number. This number must be between one and 32 inclusive.

Line 350 checks the value of the COLUMN variable. If it is not within the limits, the computer will be sent back to line 340 to get another number.

Line 360 places the column number on the screen and asks for the character code. This portion of the program will also limit the codes to zero to 255.

Line 370 checks the code value. If it is out of the limit, the computer will be directed to line 360 to get another code.

Line 380 places the character code in the command. Now it asks how many times you would like this character printed. You can print the character up to 768 times.

Line 390 checks to see how many times you want to print this character. If you requested more than 768 the computer will be sent back to line 380. If you requested zero, a one will be substituted.

Line 400 clears the question from the screen by printing a row of 38 spaces.

Line 410 places the number of times the character will be printed in the command on the screen.

Line 420 uses the VCHAR command to print the character in columns on the screen.

Line 430 asks if you want to enter another set of

codes. If you enter a "Y," the program will continue at line 310. If you enter an "N," the program will end.

CALL SCREEN

The CALL SCREEN command will change the color of the screen. We used this command in the Colors program where we changed the color and printed the color on the screen. The command for CALL SCREEN is:

CALL SCREEN (number)

The number in parentheses can be any number from one through 16. Each number has a different color assigned to it. This command is often used with the following command—CALL COLOR.

CALL COLOR

The CALL COLOR command allows you to change the color (Fig. 16-3) of any of the characters in the character set. The character set is divided into 15 different sets (Fig 16-4). When you change the color of one character in the smaller set, you change the color of all the characters in that set. In addition to changing the color of the character, you can change the background color of the character. Do not confuse the character background color with the screen color; these are two different colors.

When you are using the computer for entering programs, the background color of the character sets are set to transparent. The screen color shows through. If the background color of the character is set to any other color, you will see a box, like the cursor, around the character. You can have a yellow screen with a green background color and a red character printed inside. Because each smaller set within the character set is controled separately, you can display an entire range of different colors and color variations on the screen. The program in Listing 16-2 (flowcharted in Fig. 16-5) prints the entire character set on the screen, then cycles through various screen, character, and background colors. The second part of this program allows you to enter the number of the character set that you want to change along with the colors that you would like it to be.

Listing 16-2

Line 130 uses the RANDOMIZE command to be sure that every time the program is run the computer will choose a different number sequence. The screen is cleared and the CHAR variable is set to 31. This is the offset for the characters that will be printed on the screen. The ON WARNING NEXT is used so that if the ENTER key is

Code	Color
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
14	Magenta
15	Gray
16	White

Fig. 16-3. Color codes.

Set	Code	Characters
0	30-31	Cursor & Edge Character
1	32-39	Space, !, ", #, $, %, &, '
2	40-47	(,), *, +, comma, –, ., /
3	48-55	0, 1, 2, 3, 4, 5, 6, 7
4	56-63	8, 9, :, ;, <, =, >, ?
5	64-71	@, A, B, C, D, E, F, G
6	72-79	H, I, J, K, L, N, M, O
7	80-87	P, Q, R, S, T, U, V, W
8	88-95	X, Y, Z, , /, , , -
9	96-103	`, a, b, c, d, e, f, g
10	104-111	h, i, j, k, l, m, n, o
11	112-119	p, q, r, s, t, u, v, w
12	120-127	x, y, z, {, , }, , blank
13	128-135	undefined characters
14	136-143	undefined characters

Fig. 16-4. Character sets.

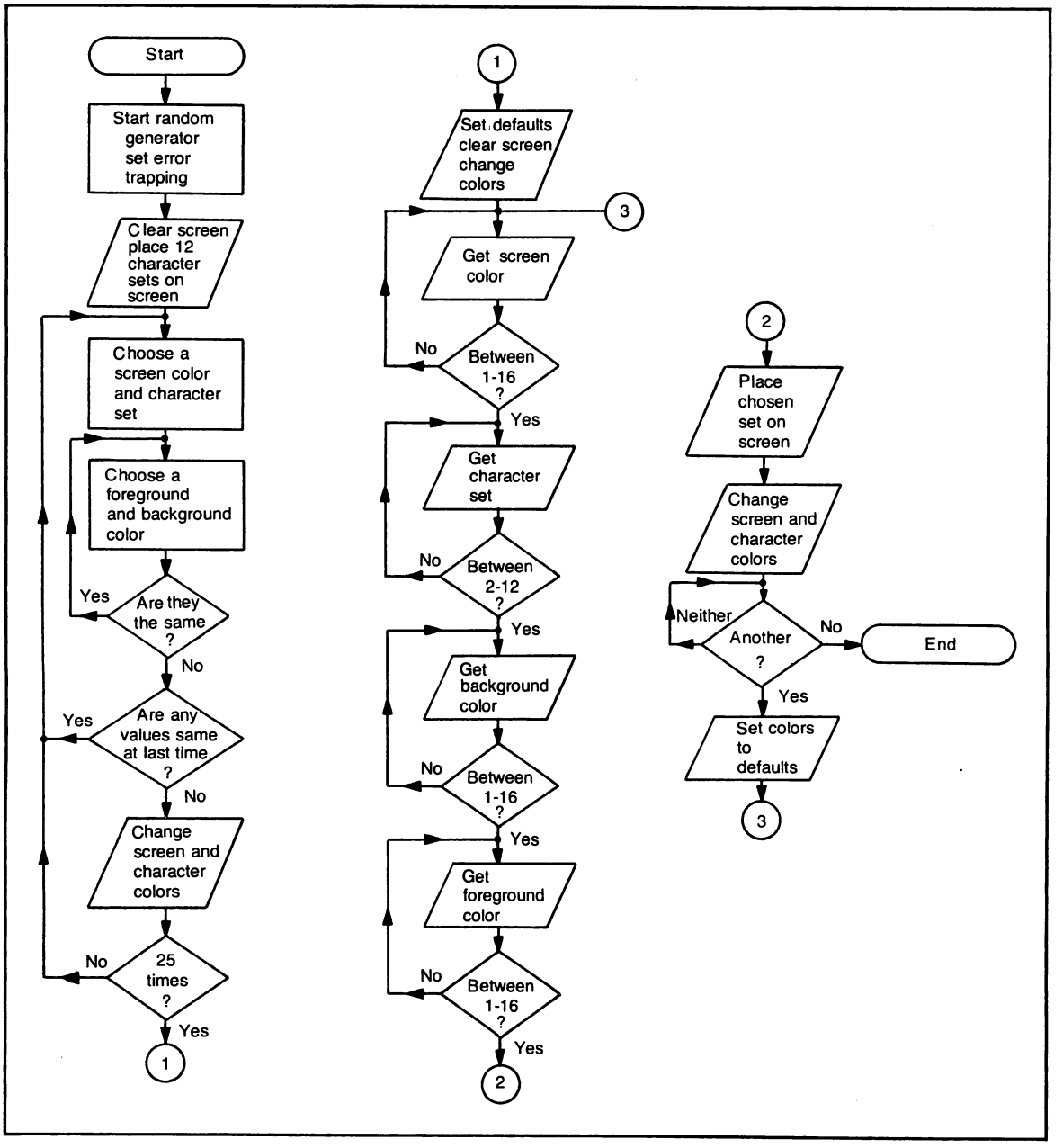


Fig. 16-5. Flowchart for Listing 16-2 Color Foreground, Background, & Screen.

pressed without entering a number, the warning message will not occur.

Line 140 begins the FOR . . . NEXT loop. This program will change the background and foreground colors for the character in the subsets one through 12. The set number will be printed on the even rows of the screen.

Line 150 is another FOR . . . NEXT loop. This time the characters for each set will be printed on the correct row of the screen. The COUNT variable

Listing 16-2

```
100 REM LISTING 16-2
110 REM COLOR FOREGROUND,BACKGROUND,& SC
REEN
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 RANDOMIZE :: CALL CLEAR :: CHAR=31 :
: ON WARNING NEXT
140 FOR ROW=2 TO 24 STEP 2 :: DISPLAY AT
(ROW,1):"SET";ROW/2
150 FOR COUNT=1 TO 8 :: DISPLAY AT(ROW,1
0+COUNT*2):CHR$(COUNT+CHAR)&" " :: NEXT
COUNT
160 CHAR=CHAR+8 :: NEXT ROW :: FOR COUNT
=1 TO 25
170 SCRN=INT(RND*16)+1 :: SET=INT(RND*12
)+2
180 BACK=INT(RND*16)+1 :: FORE=INT(RND*1
6)+1 :: IF BACK=FORE THEN 180
190 IF SCRN=LSCRN OR SET=LSET OR BACK=LB
ACK OR FORE=LFORE THEN 170
200 LSCRN=SCRN :: LSET=SET :: LBACK=BACK
 :: LFORE=FORE
210 CALL SCREEN(SCRN):: CALL COLOR(SET,F
ORE,BACK)
220 FOR DELAY=1 TO 500 :: NEXT DELAY
230 NEXT COUNT :: FOR I=2 TO 12 :: CALL
COLOR(I,2,1):: NEXT I :: CALL SCREEN(8):
: CALL CLEAR
240 DISPLAY AT(10,1):"ENTER SCREEN COLOR
(1-16)" :: ACCEPT AT(10,26)BEEP VALIDATE
(DIGIT)SIZE(2):SCRN
250 IF SCRN<1 OR SCRN>16 THEN 240
260 DISPLAY AT(12,1):"ENTER CHAR. SET(2-
12)" :: ACCEPT AT(12,23)BEEP VALIDATE(DI
GIT)SIZE(2):SET
270 IF SET<2 OR SET>12 THEN 260
280 DISPLAY AT(14,1):"ENTER BCKGRND COLO
R(1-16)" :: ACCEPT AT(14,27)BEEP VALIDAT
E(DIGIT)SIZE(2):BACK
290 IF BACK<1 OR BACK>16 THEN 280
300 DISPLAY AT(16,1):"ENTER FORGRND COLO
R(1-16)" :: ACCEPT AT(16,27)BEEP VALIDAT
E(DIGIT)SIZE(2):FORE
310 IF FORE<1 OR FORE>16 THEN 300
```

```
320 DISPLAY AT(20,1):"SET";SET
330 FOR COUNT=1 TO 8 :: DISPLAY AT(20,10
+COUNT*2):CHR$(COUNT+31+(8*(SET-1)))&" "
 :: NEXT COUNT
340 CALL SCREEN(SCRN):: CALL COLOR(SET,F
ORE,BACK)
350 DISPLAY AT(24,1):"DO ANOTHER(Y/N) ?"
 :: ACCEPT AT(24,19)BEEP VALIDATE("YN")S
IZE(1):A$ :: IF A$="" THEN 350 ELSE IF A
$="N" THEN END
360 CALL SCREEN(8):: CALL COLOR(SET,2,1)
:: GOTO 240
```

will begin with one and count to eight. There are eight characters in each subset. The value of COUNT is added to CHAR and the correct character is printed on the screen. A space is printed after every character to keep them from looking crowded.

Line 160 adds eight to the value of CHAR. We keep adding to CHAR to keep the offset correct. COUNT will only count from one to eight. If we didn't add eight to CHAR after each set of characters were printed, all the sets would have the same characters printed next to them. The FOR . . . NEXT loop continues until all the sets are numbered and the correct characters are placed next to them. Now COUNT will count from one to 25. This next part of the program will be repeated 25 times.

Line 170 chooses a random number for the screen color and stores it in the SCRN variable. One of the sets is also chosen and placed in the SET variable. We are adding two to the integer chosen because we do not want to change the color of the characters in set one or set zero. Set zero is not displayed on the screen. Set one contains the space, and we will try this program later with only one being added to the integer to see the difference.

Line 180 chooses one of the 16 colors for the background color and one for the foreground color. If the same two colors are chosen, the computer will choose two other colors. If the foreground and background colors were the same, you would not be able to see the characters in that set.

Line 190 checks all the variables to see if any of the colors or the set are the same as the one chosen the last time. We do not want the screen to stay the same color, nor do we want the same set being changed, or the same character colors being used every time. If any of the variables match the one that was used in the last cycle, the computer will choose a new set of colors.

Line 200 stores the colors that will be used in this cycle in the corresponding variables.

Line 210 uses the CALL SCREEN command to change the color of the screen and the CALL COLOR command to change the character color. The first number after the parentheses is the set that will be changed, the second variable is the foreground color, and the third variable is the background color. When this line is executed, you will see the screen change colors and one of the sets will also change. Look at the words and numbers along the left side of the screen. When the set that they belong to is changed, these characters will also change.

Line 220 is a delay loop so that the screen will pause between colors.

Line 230 continues the loop. You will see 25 changes on the screen during this loop. The next

FOR . . . NEXT loop changes the character sets back to their default values. The background color is transparent and the foreground color is black. The screen is changed to blue and cleared.

Lines 240-250 begin the second part of this program. Here you can experiment with different color combinations to see which ones are pleasing to your eyes. First you are asked for a screen color. Only a number from 1 to 16 will be accepted. This color will be stored in the SCRN variable.

Lines 260-270 ask for the character set that you would like to change the colors of. Again, you can only change the colors of sets two through 12.

Lines 280-290 ask for the background color. Enter a number between one and 16.

Lines 300-310 ask for the foreground color. Again, use only the color numbers one to 16.

Line 320 prints the set number that you chose.

Line 330 places the characters that are in that set on the screen.

Line 340 changes the screen color and the background and foreground colors of the character set that you chose.

Line 350 asks if you want to play again. Some of these characters may not show up on your screen if the colors that you chose are too close to these character colors. Enter "Y" or "N." If you entered a "Y," the screen and the character set colors would be set back to normal and the computer would be directed to line 240 where it would ask for new information. If you did not enter a "Y," the program would end.

In program line 170 we did not allow the computer to change the colors of the first set of characters. Now change that line to read as follows:

```
170  SCRN=INT(RND*16)+1::SET=INT
     (RND*12)+1
```

Run the program. Whenever the computer changes the colors of the first set, the screen also changes colors, no matter what color the computer may have chosen for it. You can see what color the screen should be by looking at the screen border. It keeps changing colors, yet the actual screen does not change colors all the time. The reason is that the space character is the first character of the first set. When the screen is cleared, spaces are printed all over the screen. Since only one character can be printed on the screen at a time, the old characters are erased. When the background color of the first character set is changed, all the spaces on the screen change to that color, so the entire screen becomes that color. The border changes to the screen color because there are no spaces printed on it.

Think of the screen as three layers; the first layer is the screen color, the second layer is the background color, and the third is the character color. In the default mode, the background color is transparent, so the screen color will show through. Once you change the background color of the first character set, it is no longer transparent, and the screen color cannot show through. You must keep this in mind when you are changing colors on the character set or you may end up with some unwanted results.

CALL GCHAR

This command allows the computer to look at a particular location on the screen. The ASCII value of the character at that location is placed in a variable. The format is:

```
CALL GCHAR(row,column,variable)
```

You specify the row and the column number. Use any variable name that you would like. The ASCII value of the specified screen location will be placed in that variable. This command is useful when you want to check to see if a character is in a particular place, such as in a grid game where the program needs to know which playing piece is in which square. In the program in Listing 16-3 (flowcharted in Fig. 16-6), you are to follow the path on the screen without running into the walls. The GCHAR command is used before the next part of your path is placed on the screen to make sure that your path is not being placed on the wall. Use the arrow

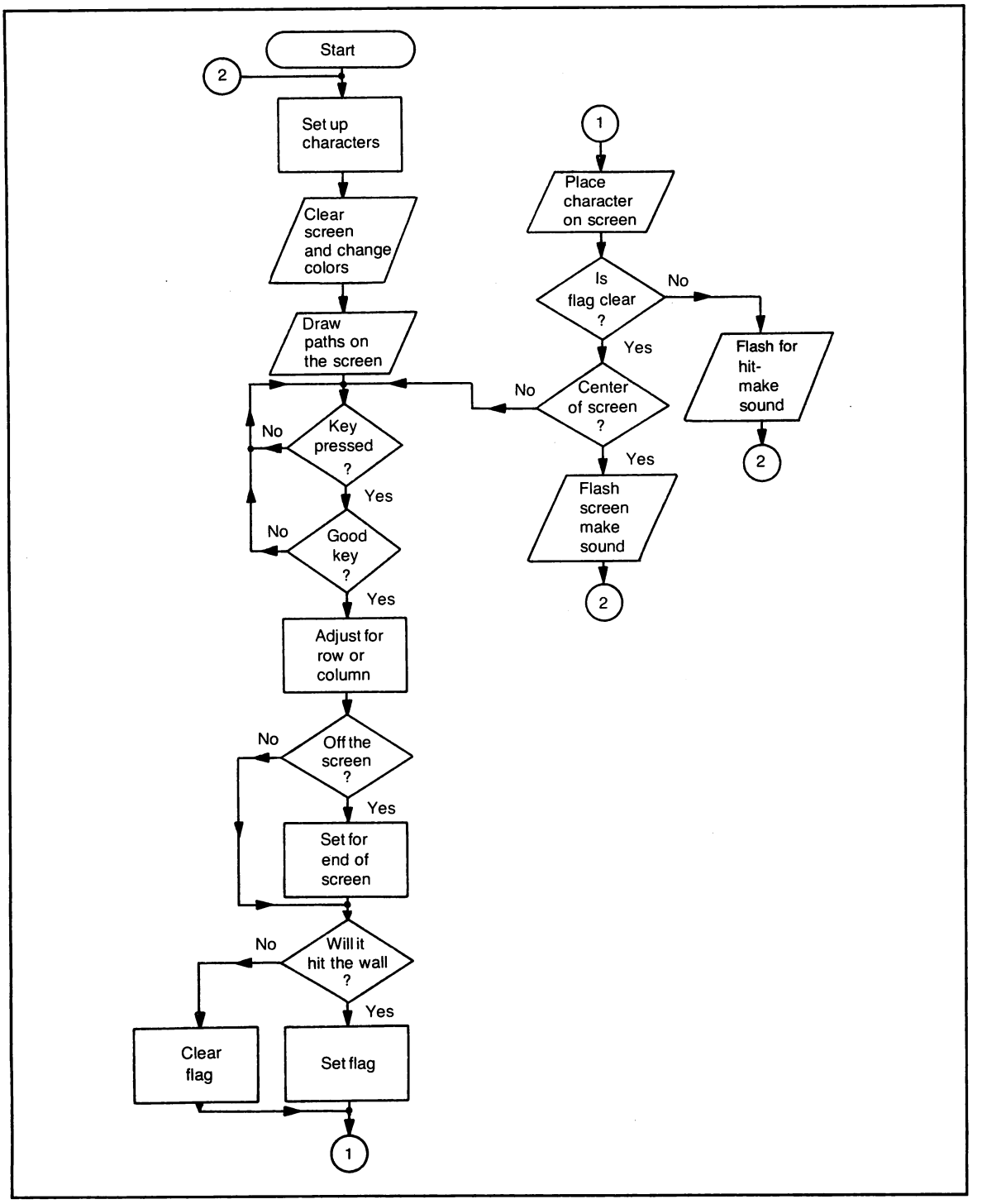


Fig. 16-6. Flowchart for Listing 16-3 Paths.

Listing 16-3

```
100 REM LISTING 16-3
110 REM PATHS
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 C$="FFFFFFFFFFFFFFFF" :: CALL CHAR(1
43,C$,135,C$):: CALL COLOR(14,6,1,13,16,
1)
140 RESTORE 160 :: CALL CLEAR :: CALL SC
REEN(2)
150 FOR CT=1 TO 7 :: READ R,C,RPT :: CAL
L HCHAR(R,C,143,RPT):: NEXT CT
160 DATA 2,3,26,6,11,14,9,15,7,11,17,3,1
7,17,5,20,15,10,24,11,18
170 FOR CT=1 TO 7 :: READ R,C,RPT :: CAL
L VCHAR(R,C,143,RPT):: NEXT CT
180 DATA 6,11,19,9,15,12,11,17,7,11,19,5
,9,21,9,6,24,15,2,28,23
190 R=20 :: C=6 :: CALL HCHAR(R,C,135)
200 CALL KEY(0,K,S):: IF S=0 THEN 200
210 IF K=68 OR K=100 THEN C=C+1 :: GOTO
260
220 IF K=69 OR K=101 THEN R=R-1 :: GOTO
260
230 IF K=83 OR K=115 THEN C=C-1 :: GOTO
260
240 IF K=88 OR K=120 THEN R=R+1 :: GOTO
260
250 GOTO 200
260 IF R<1 THEN R=1 ELSE IF R>24 THEN R=
24
270 IF C<3 THEN C=3 ELSE IF C>28 THEN C=
28
280 CALL GCHAR(R,C,T):: IF T=143 THEN F=
1 ELSE F=0
290 CALL HCHAR(R,C,135)
300 IF F=0 THEN IF R=12 AND C=18 THEN 34
0 ELSE 200
310 CALL SOUND(250,110,5,110,5,200,5,-8,
5)
320 FOR C=1 TO 3 :: CALL SCREEN(14):: CA
LL SOUND(250,110,5,110,5,200,5,-8,5):: C
ALL SCREEN(10)
330 FOR DELAY=1 TO 100 :: NEXT DELAY ::
NEXT C :: GOTO 130
```

```
340 CALL COLOR(14,5,1,13,7,1)!! GOSUB 36
0 !! CALL COLOR(14,8,1,13,13,1)!! GOSUB
360 !! CALL COLOR(14,11,1,13,6,1)!! GOSU
B 360
350 GOTO 130
360 FOR C=1 TO 4 !! CALL SOUND(200,110*C
,5,147*C,5,185*C,5)!! NEXT C !! RETURN
```

keys—S, D, X, and E without the function key to make the path.

Listing 16-3

Line 130 creates a new character. The code is placed into C$. The CALL CHAR command places this character code into the 143rd character of the character set. It also places that same character code into the 135th character of the set. The CALL COLOR command changes the color of the 14th set to light blue and the color of the 13th set to white.

Line 140 sets the pointer to line 160. When the data is read, the computer will begin at this line number. This is not necessary the first time that the program is run, but when the program is repeated without the RUN command, the computer needs to know where the data is. The screen is cleared and the screen color is set to black.

Line 150 is a FOR . . . NEXT loop. The computer reads the row and column number and the number of times the character will be printed from the DATA line that follows. The HCHAR command is used to draw the horizontal lines of the maze on the screen. The loop continues until all the lines are drawn.

Line 170 is similar. This time the computer draws the vertical lines of the maze using the information from the DATA line that follows.

Line 190 sets the starting row and column for your cursor and positions it on the screen.

Line 200 uses the CALL KEY command to find out which key is pressed. If the value of the S variable is zero, then no key has been pressed. The computer will remain at this line until a key has been pressed. The ASCII value of the key will be stored in the K variable.

Lines 210-240 check the value of the K variable. If the value is 68 or 100 then the D key has been pressed. One is added to the value of C so that the cursor can be moved one column to the right. If the value of K is 69 or 101, then the E key has been pressed. One is subtracted from the value of R so that the cursor can move up one row on the screen. When the value of K is 83, the S key has been pressed. To move the cursor to the left, one is subtracted from the value of C. The K variable has the value of 88 when the X key has been pressed. One is added to the value of R to move the cursor down one row on the screen. After the row or column variable is adjusted, the computer is sent to line 260. By checking the K variable for two values, you do not have to worry about the ALPHA LOCK key. The first value is the uppercase value, the second the lowercase. The program will work correctly whether the ALPHA LOCK key is up or down.

Line 250 is executed if the key that was pressed was not one of the arrow keys. The computer will loop until a correct key is pressed.

Line 260 checks the value of the R variable. If the value is less than one, or greater than 24, the cursor would be printed off the screen. The variable is reset to the edge value.

Line 270 checks the value of the column variable—C. If this value is less than 3 or greater than 28, then the cursor would be off the screen, so the program resets the variable to the correct edge value.

Line 290 uses the GCHAR command to find out

what is on the screen at the location that the cursor will be printed at. At this time, the cursor has not moved. Before the computer will place the cursor at the new location on the screen, it needs to know what is currently on the screen at that position. The ASCII value of that location will be placed in the T variable. If the value of T is 143, then the wall is on the screen at that location. The F variable is used as a flag. If the cursor will hit the wall, the variable is set to one for true; otherwise it is set to zero.

Line 290 places the cursor on the screen at the new location. Character 135 is the cursor.

Line 300 checks the value of F. If it is a zero, the computer checks the row and column that the cursor is in. If the cursor makes it to the twelfth row and eighteenth column without hitting the wall, the computer is sent to line 340. Otherwise it is sent to line 200.

Line 310 is used if the value of F is a one. The CALL SOUND is used to call your attention to the fact that you hit a wall.

Lines 320-330 flash the screen and continue the sound. After it has finished, the computer is sent to line 130 to start another game.

Lines 340-360 are used when the cursor reaches the center of the maze. The screen flashes and the computer sounds to let you know you have made it without hitting any of the walls.

USING SOUND COMMANDS

CALL SOUND

Your TI-99/4A computer is equipped with four separate and distinct sound generators. Each voice or sound generator is capable of producing its own tone or noise independent of the other generators. Only three tones may be produced at one time. The fourth generator can produce noise while the other three are producing tones. The format for producing tones or noise is as follows:

CALL SOUND(duration,frequency,volume)

The first variable sets the duration or length of time that the computer will produce the tone. The number is in thousandths of a second, so if you want to produce a tone for one second, the first number would have to be 1000. Two seconds would be 2000, and a quarter of a second 250. If the value of this variable is positive, the computer will not make the sound until the previous sound has been completed. If, however, the value of the variable is negative, the computer will begin the new sound immediately.

The second variable is the frequency of the tone. The frequency of a tone is the actual tone itself. Every note or tone vibrates at a different frequency. The larger the frequency, the higher the tone; the smaller the frequency, the lower the tone. If the variable contains a position number such as 110, 440, or 1047, the sound will be a tone. The tone values are any positive number between 110 and 44733. A negative number will produce noise. The noise values are the negative numbers between negative one and negative eight.

The third value is the volume. This value determines how loud the sound will be. The value can be any positive number between zero and 30 with zero being the loudest and 30 the softest. A negative number cannot be used for the volume.

The program in Listing 16-4 (flowcharted in Fig. 16-7) demonstrates using the SOUND command with negative for the duration. In this program the computer will produce only one tone at a time.

Listing 16-4

Line 130 creates new characters. C$ has been set to a character pattern, which was the character numbers 143, 135, and 127. On this line the pattern in C$ is changed. The new pattern is used for character numbers 142, 134, and 125.

Line 140 clears the screen and sets the screen color to cyan. The characters in the fourteenth set are changed to white on a transparent background; the thirteenth set is changed to light blue on a transparent background, and the twelfth set is changed to black on a white background. These three sets will be used for the piano keys.

Line 150 prints the title of the program on the screen.

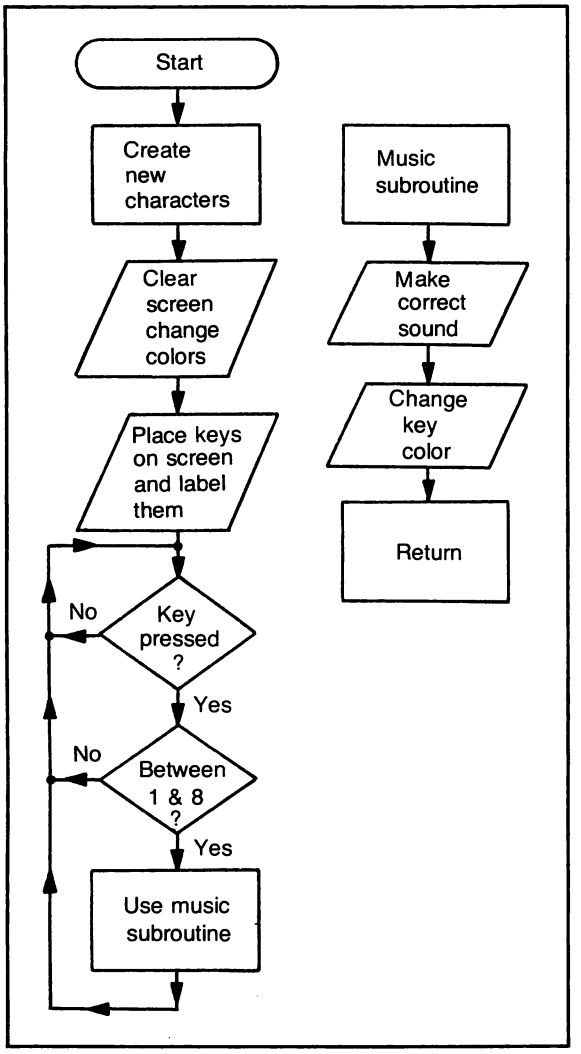


Fig. 16-7. Flowchart for Listing 16-4 Play a Tune.

Line 160 is a FOR . . . NEXT loop that places the piano keys on the screen. The data in line 170 tells the computer which column the key should be drawn in. The value of CT sends the computer to the subroutine that draws the correct key.

Line 180 is another FOR . . . NEXT loop. This routine draws in the black key. The data in line 190 tells the computer in which row the black key begins. The VCHAR command is used to place the key on the screen.

Line 200 places the letter names of the keys on the screen.

Line 210 uses the CALL KEY command to find out which key has been pressed. If the value of the S variable is a zero, then a key has not been pressed and the computer loops at this location until a key has been pressed. When the value of S is not zero, the value of the K variable is checked. If its value is less than 49 or greater than 56, a number key between one and eight has not been pressed. The computer will loop back to the beginning of this line. The computer will remain at this line until a valid character has been entered.

Line 220 subtracts 48 from the value of K. The ASCII value of the number one is 49. By subtracting 48 from the ASCII value, we arrive at the number of the key that has been pressed. The computer is directed to the correct sound line based on the value of K.

Line 230 uses the CALL KEY command again. This time the S variable is checked for a negative one. If the value of S is a negative one, then the same key was pressed. The value of S will be positive if a new key is pressed, and a zero if no key is

Listing 16-4

```
100 REM LISTING 16-4
110 REM PLAY A TUNE
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 C$="FFFFFFFFFFFFFFFF" :: CALL CHAR(1
43,C$,135,C$,127,C$):: C$="0101010101010
101" :: CALL CHAR(142,C$,134,C$,125,C$)
140 CALL CLEAR :: CALL SCREEN(8):: CALL
```

```
COLOR(14,16,2,13,6,2,12,2,16)
150 DISPLAY AT(6,8):"PLAY A TUNE"
160 FOR CT=1 TO 8 :: READ C :: ON CT GOS
UB 350,380,400,350,380,380,400,350 :: NE
XT CT
170 DATA 5,8,11,14,17,20,23,26
180 FOR CT=1 TO 11 :: READ C :: CALL VCH
AR(14,C,127,4):: NEXT CT
190 DATA 7,8,10,11,16,17,19,20,22,23,28
200 DISPLAY AT(22,4):"C  D  E  F  G  A
B  C"
210 CALL KEY(0,K,S):: IF S=0 THEN 210 EL
SE IF K<49 OR K>56 THEN 210
220 ON K-48 GOSUB 260,270,280,290,300,31
0,320,330
230 CALL KEY(0,KY,S):: IF S=-1 AND KY=K
THEN 220
240 ON K-48 GOSUB 350,380,400,350,380,38
0,400,350
250 GOTO 210
260 CALL SOUND(-250,131,5):: C=5 :: GOTO
 340
270 CALL SOUND(-250,147,5):: C=8 :: GOTO
 370
280 CALL SOUND(-250,165,5):: C=11 :: GOT
O 390
290 CALL SOUND(-250,175,5):: C=14 :: GOT
O 340
300 CALL SOUND(-250,196,5):: C=17 :: GOT
O 370
310 CALL SOUND(-250,220,5):: C=20 :: GOT
O 370
320 CALL SOUND(-250,247,5):: C=23 :: GOT
O 390
330 CALL SOUND(-250,262,5):: C=26 :: GOT
O 340
340 CALL VCHAR(14,C,135,7):: CALL VCHAR(
14,C+1,135,7):: CALL VCHAR(18,C+2,135,3)
:: RETURN
350 CALL VCHAR(14,C,143,7):: CALL VCHAR(
14,C+1,143,7):: IF C+2<28 THEN CALL VCHA
R(18,C+2,125,3):: RETURN
360 CALL VCHAR(18,C+2,143,3):: RETURN
```

```
370 CALL VCHAR(18,C,135,3)!! CALL VCHAR(
14,C+1,135,7)!! CALL VCHAR(18,C+2,135,3)
!! RETURN
380 CALL VCHAR(18,C,143,3)!! CALL VCHAR(
14,C+1,143,7)!! CALL VCHAR(18,C+2,125,3)
!! RETURN
390 CALL VCHAR(18,C,135,3)!! CALL VCHAR(
14,C+1,135,7)!! CALL VCHAR(14,C+2,135,7)
!! RETURN
400 CALL VCHAR(18,C,143,3)!! CALL VCHAR(
14,C+1,143,7)!! CALL VCHAR(14,C+2,125,7)
!! RETURN
```

pressed. If the value of S is a negative one and the ASCII value of the key is the same as the last variable, the computer is sent back to line 220 to continue the sound.

Line 240 is used when a new key is being pressed, or no key is being pressed. The value of the K variable is the ASCII value of the key, so 48 is subtracted from it so the computer can use the actual value of the key. The computer is sent to the correct subroutine based on this new value.

Line 250 sends the computer back to line 210 to wait for another key to be pressed.

Lines 260-330 contain the SOUND commands. The computer will be sent to one of these lines based on the value of K. The value of the duration is negative. This means that the computer will use the new values as soon as it receives this command. The tone will be played for a quarter of a second. The value of the tone is based on the note values for low C to middle C. The sound will be relatively loud. After the computer makes the correct sound, the C variable is set to a value. This value is the column of the key on the screen that is producing the note. The computer is sent to the correct subroutine.

Lines 340-400 display the piano keys on the screen. The lines are in sets of two. The first line places the blue key on the screen. The second line restores the key to white. This way, when you press a key to create a sound, the screen will indicate which key is being pressed. The tone will continue and the key will remain colored until a new key is pressed or no keys are pressed.

In program lines 260 through 300 the duration is a negative number. Change it to a positive number and run the program. Hold a key down for a period of time. You will hear the tone pulsating. When the duration is positive, the computer does not use the new SOUND command until the one that it is using has completed its cycle and the sound generator shuts off. Then the computer uses the SOUND command again, turning the sound generator on for the length of time specified by the duration. Replace the values with the negative ones. Now run the program and hold a key down. The tone is steady. With the negative value, the computer changes to the new SOUND command immediately. The sound generator does not shut off and on again, so there is no pulsing effect, just smooth tones.

The program in Listing 16-5 (flowcharted in Fig. 16-8) combines some simple graphics with two part harmony. Your TI-99/4A computer is capable of three part harmony. In this program, we will use two sound generators at the same time.

Listing 16-5

Line 130 clears the screen so that you do not see the characters being changed if any are on the screen.

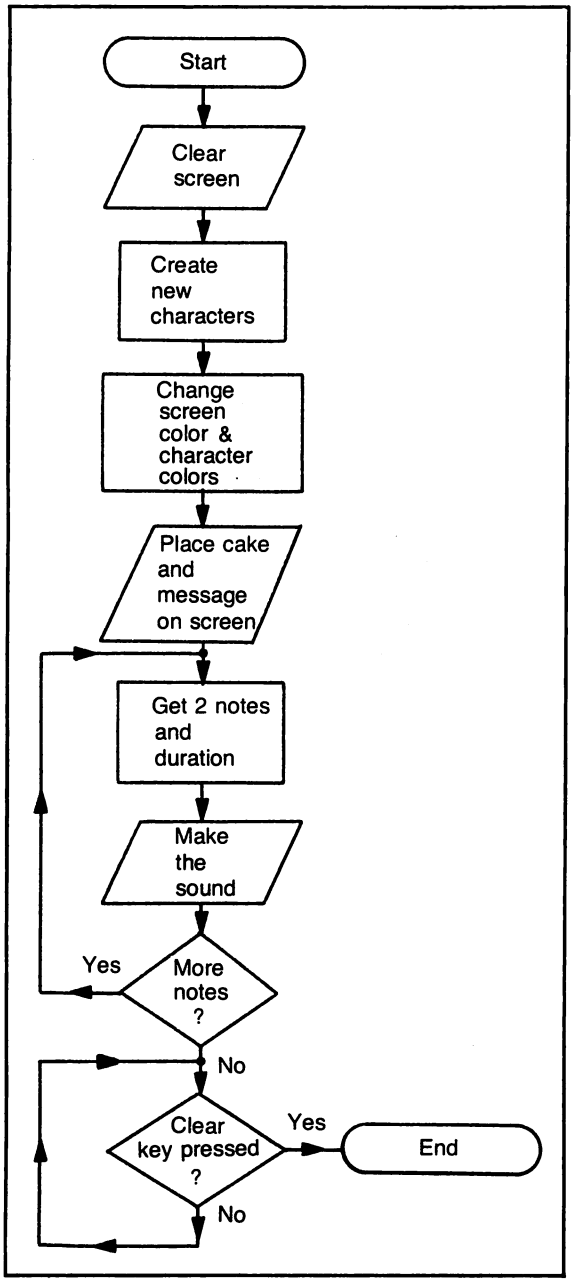


Fig. 16-8. Flowchart for Listing 16-5 Cake.

The characters from ASCII 99 through 109 are changed. The character patterns for these characters begin with line 250. (See the altered characters in Fig. 16-9.)

Line 140 changes the characters from 112 to 116. We miss the characters between 109 and 112 because we want these to be a different color, so we are starting at a new character set.

Line 150 changes two more characters. These will be the candles on the top of the cake.

Line 160 changes some of the uppercase letters. Since we are not changing all of the uppercase letters, the computer reads a number before it reads the character pattern. This number is the ASCII value of the character that will be changed.

Line 170 changes the colors in the character sets 9, 10, and 11. In set 9, the foreground or character color is the same as the screen color. The background is set to light red. This eliminates code to enter for the character set because most of the character will be light red. The same thing holds true for set 10. The background or character color is set to medium red. We use this color for the edge so you can see the difference between the top and the side of the cake. The background color is light red. The last set, 11, is the plate. The foreground or character color is medium green and the background color is transparent so that the screen color will show through.

Lines 180-200 place the cake with the lit candles on the screen.

Line 210 prints HAPPY BIRTHDAY on the screen These letters have been changed to a fancier character set.

Line 220 plays the melody. There are 26 sets of notes in the song. The computer reads in the duration or length of the note (D variable), the first note (N) and the second note (N1). The SOUND command plays both notes at the same time. The value for the duration is the first variable followed by the first note and its volume. The

Listing 16-5

```
100 REM LISTING 16-5
110 REM CAKE
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
```

```
130 CALL CLEAR :: FOR I=99 TO 109 :: REA
D C$ :: CALL CHAR(I,C$):: NEXT I
140 FOR I=112 TO 116 :: READ C$ :: CALL
CHAR(I,C$):: NEXT I
150 FOR I=120 TO 121 :: READ C$ :: CALL
CHAR(I,C$):: NEXT I
160 FOR I=1 TO 9 :: READ C,C$ :: CALL CH
AR(C,C$):: NEXT I
170 CALL COLOR(9,12,10,10,9,10,11,3,1)::
 CALL SCREEN(12)
180 DISPLAY AT(10,12):"deccfs" :: DISPLA
Y AT(11,12):"hiiiiJ" :: DISPLAY AT(12,12
):"111111"
190 DISPLAY AT(13,11):"pk1111mt" :: DISP
LAY AT(14,11):"qrrrrrrs"
200 DISPLAY AT(9,14):"xx"
210 DISPLAY AT(6,11):"H A P P Y" :: DISP
LAY AT(17,8):"B I R T H D A Y"
220 FOR I=1 TO 26 :: READ D,N,N1 :: CALL
 SOUND(D,N,0,N1,0):: NEXT I
230 DISPLAY AT(9,14):"yy"
240 GOTO 240
250 DATA FF,FFFFFCF0C08,FF8,FF01,FFFF3F0
F0301
260 DATA 40201F,0000FF,0204FC,0000000000
80C0E,00,0000000000010307
270 DATA 0000003070F0F07,0301,FFFFFF,C0
8,000000C0E0F0F0E,0010029012929292,00000
01012929292
280 DATA 65,7E66667E6666F7,66,FC66667C66
66FC,68,FC6666666666FC,72,F766667E6666F7
290 DATA 73,3C18181818183C,80,FC66667C60
60F,82,FC66667C666677,84,7E5A5A1818183C
300 DATA 89,EF6666667E067E
310 DATA 250,262,220,250,262,220,500,294
,220,500,262,220,500,349,262,750,330,262
320 DATA 250,262,196,250,262,196,500,294
,196,500,262,196,500,392,262,750,349,262
330 DATA 250,262,220,250,262,220,500,523
,349,500,440,349,250,349,262,250,349,262
,500,330,262,750,294,294
340 DATA 250,466,349,250,466,349,500,440
,349,500,349,262,500,392,262,750,349,262
```

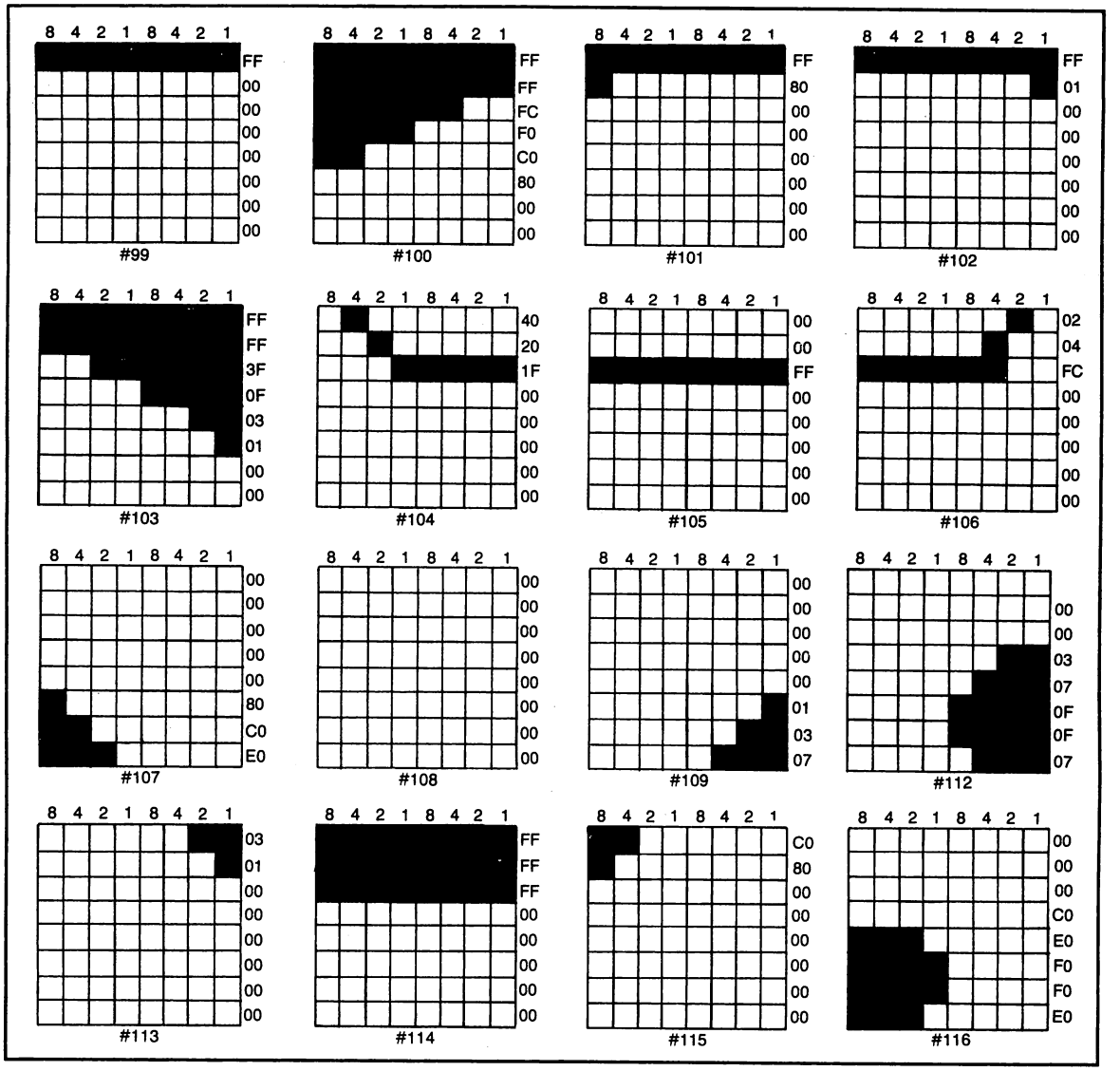


Fig. 16-9. Characters for Listing 16-5 Cake.

second note and its volume follows. The duration for both notes is the same. The loop continues until all the notes have been played. The melody is located in lines 310 through 340. The duration for all the notes is a positive number. This means that the next note will not sound until the computer has finished playing the current note.

Line 230 blows out the candles after the song. Actually, because the computer will continue with the program before the last note has finished, the candles will be blown out during the last note.

Line 240 keeps the program from ending until the CLEAR key has been pressed.

MIXING GRAPHICS AND SOUND

The program in Listing 16-6 (flowcharted in Fig. 16-10) combines graphics with sound. It is an example of the classic pencil and paper game—

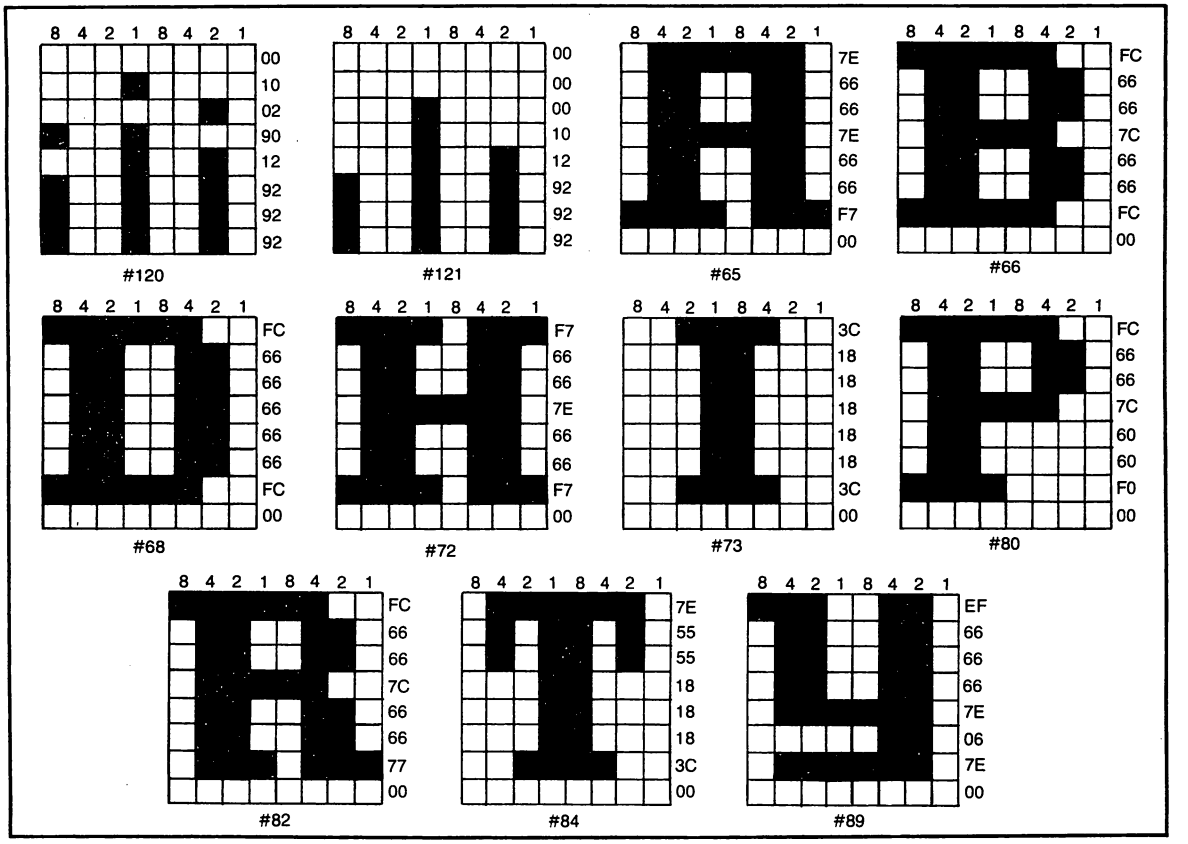


Fig. 16-9. Characters for Listing 16-5 Cake. (Continued from page 173.)

battleships. But this time, you are playing against the computer.

Listing 16-6

Line 130 uses the RANDOMIZE command to ensure that every game will be different.

Line 140 is a FOR . . . NEXT loop that reads the length of each ship into an array. The computer will use this information when it is placing the ships in its grid.

Line 150 is the data for the ships. Each player has ten ships to place in the grid that are four different sizes.

Line 160 places the character number 103 in three string arrays. These arrays will be used to determine where the ships hits, and misses are. At this time, character 103 is undefined. We will create this character in the next routine.

Line 170 places the data pointer at line 200, where the information for creating the new characters is stored. The screen clears, and the computer reads the character patterns from the DATA line. The new characters are placed in locations 124 through 143 (Fig. 16-11).

Line 180 sets the C variable to 110. This is the first location of the character that will be created. The FOR . . . NEXT loop counts from 134 to 143. The character pattern from these locations is placed in C$. Then that pattern is transferred to the location set by C. The C variable is incremented by one and the loop continues. The characters are transferred from one location to the other. The characters are the same; however, when they appear on the screen, they will be in two different colors. By transferring them to another set, the

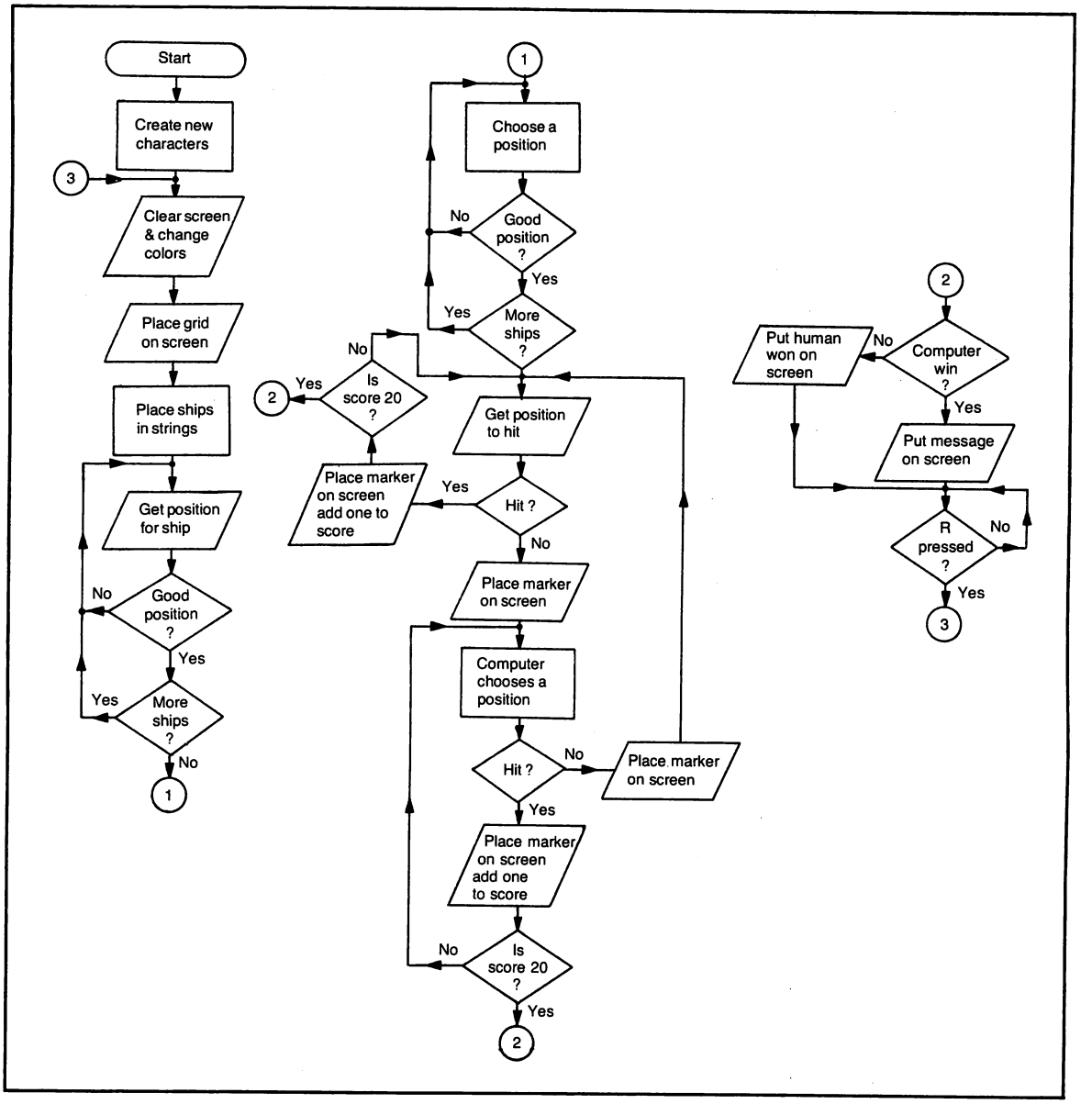


Fig. 16-10. Flowchart for Listing 16-6 Battleship.

Listing 16-6

```
100 REM LISTING 16-6
110 REM BATTLESHIP
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 RANDOMIZE
```

```
140 FOR C=1 TO 10 :: READ L :: SHP(C)=L
:: NEXT C
150 DATA 4,3,3,2,2,2,1,1,1,1
160 FOR C=1 TO 10 :: H$(C)=RPT$(CHR$(103
),10):: COM$(C)=RPT$(CHR$(103),10):: SCR
$(C)=RPT$(CHR$(103),10):: NEXT C
170 RESTORE 200 :: CALL CLEAR :: FOR C=1
24 TO 143 :: READ C$ :: CALL CHAR(C,C$):
: NEXT C
180 C=110 :: FOR I=134 TO 143 :: CALL CH
ARPAT(I,C$):: CALL CHAR(C,C$):: C=C+1 ::
 NEXT I
190 FOR C=101 TO 103 :: READ C$ :: CALL
CHAR(C,C$):: NEXT C
200 DATA 000103031F130301,000A030F0F0303
0F,1F0F03030B0F0B02,00080C0E7F5F5F7F,0F3
F3F1F1F0F0F0F,1F1F3F2F2F0E0C08
210 DATA 000416171F0F070F,071F070F07070F
3F,7F3F0F071F07070F,1F171707070F0604,000
0003010107CFE,00000009E98FEFC
220 DATA 000000017323FF7F,00F090F0FEFCF8
F0,0000061EFFFFFFFF,00001C07FF7F3F1F,000
0001CB0FEFCF8,00008082CAFFFFFF
230 DATA 0001031397FFFFFF,0000000721FF7F
3F,FFC3A59999A5C3FF,FFC399BDBD99C3FF,FF8
181818181FF
240 HS,CS,CHR=0
250 CALL SCREEN(1):: CALL COLOR(14,5,1,1
3,5,1,12,5,1,11,16,1,10,16,1,9,16,1)
260 FOR C=0 TO 8 :: CALL COLOR(C,10,1)::
 NEXT C
270 DISPLAY AT(2,1):"COMPUTER" :: DISPLA
Y AT(2,24):"HUMAN" :: DISPLAY AT(7,10):"
ABCDEFGHIJ"
280 GOSUB 1470 ! PUT GRID ON SCREEN
290 BV$=CHR$(133)&CHR$(132)&CHR$(131)&CH
R$(130):: CV$=CHR$(129)&CHR$(128)&CHR$(1
27):: DV$=CHR$(126)&CHR$(125)
300 SBV$=CHR$(124):: SV$(1)=BV$ :: SV$(2
)=CV$ :: SV$(3)=CV$ :: FOR C=4 TO 6 :: S
V$(C)=DV$ :: NEXT C
310 FOR C=7 TO 10 :: SV$(C)=SBV$ :: NEXT
 C
```

continued on page 178

colors can be changed and both sets can be printed on the screen at the same time and in two different colors.

Line 190 places the last three characters in locations 101 to 103.

Line 240 sets the HS, CS, and CHF variables to zero. The first two variables keep track of the player's score and the computer's score. The third variable is set when the computer has hit a ship.

Line 250 changes the color of the screen and the character sets 9 through 14.

Line 260 is a FOR . . . NEXT loop to change the color in the first nine sets.

Line 270 displays the players' names and the letters for the grid.

Line 280 sends the computer to a subroutine that places the grid on the screen.

Lines 290-340 place the characters for the ships in string arrays. There are two sets of ships that will be used in the program. Both sets are placed in string arrays. The ships with the "V" in the variable name will be vertical on the screen. The ships with the "H" will be horizontal.

Line 350 places all four ships on the screen. The ship that you will be placing in the grid is a different color.

Line 360 begins the FOR. . . NEXT loop that places the ten ships on the grid. You will be placing one battleship, two cruisers, three destroyers, and four submarines on the grid.

Line 370 sets the flag variable to zero. This variable will be set to one if the position where the ship is being entered is invalid, or the X key was pressed to delete or erase the ship from the screen. The first subroutine at line 740 gets the letter and number of the position of the ship. The second subroutine at line 870 finds out if you want the ship going across or up on the grid. It also checks the position to make sure that the ship can fit in that position and waits for the ENTER key or an X. If the ENTER key is pressed, then the ship is in the position that you want it in. If the X key is pressed, the ship is erased, and the FLAG variable is set to one. The line will repeat itself until the FLAG variable is zero.

Line 380 continues the loop until all the ships are placed on the grid. Then the computer will tell you that it is placing its ships on the grid.

Line 390 begins the computer's FOR . . . NEXT loop to place the ships on the grid.

Line 400 sets the flag to zero again. The computer chooses two random numbers, one for the letter and one for the number.

Line 410 chooses a zero or a one. If the computer chooses a one, the ships will be placed across on the grid, otherwise it will be placed up.

Line 420 uses the subroutine at line 1280 to see if the computer can place the ship at that location. If it cannot, the flag will be set to one and the computer will go to line 400 to choose a new location. If the computer can place the ship there, the loop will continue.

Line 430 continues the loop until all the ships have been placed onto the computer's grid.

Line 450 begins the game. The computer uses the subroutine at line 1470 to place the elements of SCR$ on the screen. This string array stores the screen information. If there was a hit or miss, it will be in this string and then placed on the screen. The color of the characters are changed. The same characters are used for the computer grid and the player grid. By changing the color of the character set, we know whose turn it is.

Line 460 uses the subroutine at line 740 to get the letter and number of a square on the grid. The computer looks at that location in its string array to see if it was used before. The characters 101 and 102 are used to indicate a hit or a miss. If that location was used before, the computer goes back to the beginning of this line and the player must try another location. Each location can be entered only once in a game.

Line 470 checks to see if that location was a 103. If it was not, the player hit a ship and the computer is sent to line 660 to flash the screen, tally the score, and place the hit marker on the grid.

Line 480 places the miss marker in a temporary string and uses the subroutine at line 1490 to place the marker on the screen and in the correct string arrays for storage.

Line 500 begins the computer's turn. The grid color

```
320 BH$=CHR$(143)&CHR$(142)&CHR$(141)&CH
R$(140)!! CH$=CHR$(139)&CHR$(138)&CHR$(1
37)!! DH$=CHR$(136)&CHR$(135)
330 SBH$=CHR$(134)!! SH$(1)=BH$ !! SH$(2
)=CH$ !! SH$(3)=CH$ !! FOR C=4 TO 6 !! S
H$(C)=DH$ !! NEXT C
340 FOR C=7 TO 10 !! SH$(C)=SBH$ !! NEXT
 C
350 DISPLAY AT(19,8)!BH$&" "&CHR$(115)&C
HR$(114)&CHR$(113)&" "&CHR$(112)&CHR$(11
1)&" "&CHR$(110)
360 FOR SH=1 TO 10
370 FLAG=0 !! GOSUB 740 !! GOSUB 870 !!
IF FLAG THEN 370
380 NEXT SH !! DISPLAY AT(21,1)!TAB(6)!"
PLACING MY SHIPS"
390 FOR SH=1 TO 10
400 FLAG=0 !! SC=INT(RND*10)+1 !! N=INT(
RND*10)+1
410 K=INT(RND*2)!! IF K<>0 AND K<>1 THEN
 410 ELSE IF K=1 THEN K=65 ELSE K=85
420 GOSUB 1280 !! IF FLAG THEN 400
430 NEXT SH
440 REM HUMAN'S TURN
450 GOSUB 1470 !! CALL COLOR(9,5,1)
460 GOSUB 740 !! T=ASC(SEG$(COM$(N),SC,1
))!! IF T=101 OR T=102 THEN 460
470 IF T<>103 THEN 660 ! HIT
480 TEMP$=CHR$(102)!! GOSUB 1490
490 REM COMPUTER'S TURN
500 CALL COLOR(9,16,1)!! FOR C=1 TO 10 !
! DISPLAY AT(7+C,10)SIZE(10)!H$(C)!! NEX
T C
510 DISPLAY AT(21,1)!"      HIT ON POSITI
ON"
520 IF CHF=0 THEN 590
530 FOR R=1 TO 10 !! FOR C=1 TO 10 !! T=
ASC(SEG$(H$(R),C,1))!! IF T<>101 THEN 58
0
540 IF C<>10 THEN T1=ASC(SEG$(H$(R),C+1,
1))!! IF T1<>101 AND T1<>102 THEN SC=C+1
 !! N=R !! T=T1 !! GOTO 600
550 IF C<>1 THEN T1=ASC(SEG$(H$(R),C-1,1
```

continued on page 182

is changed to white and the player's grid is placed on the screen. This grid contains the ships that the player placed on their grid.

Line 510 displays a message on the screen.

Line 520 checks the CHF variable. This variable is used as a flag. If the computer hit a ship the last time it had a turn, this variable will be set to one and the computer will use the routine to find the hit and try another location near it. If the computer did not get a hit the last time, or has already tried all the locations near the last hit, this variable will be set to zero and the computer will go on to line 590.

Lines 530-580 check every location in the grid for the hit. When it finds the hit location, it checks the location after the hit, then before the hit, then below the hit, then above the hit. If any of these locations have not been used, that is, they do not contain a hit or miss marker in them, the computer will hit them. It does not check these locations to see if there is a ship in them—that would be cheating. It only checks to see if it can drop a hit there. If it can, the computer goes on to line 600 otherwise it finishes the loop. If the computer completes the loop and does not find a location to hit, the CHF variable is set to zero and the computer continues with line 590.

Line 590 is used when the computer does not have a specific location to hit. The computer chooses a random position on the grid, then checks it to see if it has been used before. If it cannot use the location, it tries again until it finds a location that has not been tried before.

Line 600 prints the location that will be hit on the screen. The SC variable has 64 added to it so that the letter will be printed.

Line 610 checks the location that has been hit. If it is not character 103, then a ship has been hit, and the computer goes on to line 630.

Line 620 places the miss marker in the temporary string and uses the subroutine at line 1630 to display it on the screen and change the characters in the string array to reflect the miss. The computer goes back to line 450 for the player's turn.

Line 630 uses the subroutine at line 1610 to flash the screen. The hit marker is placed in the temporary string; one is added to the computer's score; the subroutine at line 1630 is used to place the hit on the grid, and the CHF variable is set to one.

Line 640 checks the score. If the computer has not scored 20 points, the computer goes to line 510 to try again. If the score is 20, the computer goes to line 680 to end the game.

Line 660 is used when the player gets a hit. The subroutine at line 1610 is used to flash the screen; the hit marker is stored in the temporary string, and the player's score is increased by one. The computer then goes to the subroutine at line 1490 to place the marker on the screen and in the appropriate string arrays.

Line 670 checks the player's score and if it is less than 20, the computer will go to line 460 to give you another turn. If the player's score is 20, the computer will continue with the next line.

Line 680 checks the computer's score. If it is 20, the computer wins, and the message is printed on the screen.

Line 690 checks the player's score; if it is 20, then the player wins and that news is flashed on the screen.

Line 700 tells you to press the R key to play again.

Line 710 waits for a key to be pressed. As long as S is zero, the computer will loop back to the beginning of this line. When S is not zero, the K variable is checked for 82—the R key. If it is not the R key, the computer will loop back to the beginning of this line.

Line 720 sends the computer to line 160 to play another game.

Line 740 begins the routine that gets the letter and number of the square on the grid from the player. First, the message is printed on the screen.

Line 750 waits for a key to be pressed. Once a key is pressed, the K variable is checked to see if it is a letter. If the ASCII value of the key is less than 65 or greater than 74, the key is not a letter and the computer remains at that line until a letter key is pressed.

Line 760 subtracts 64 from the value of K. This number will be the grid column of the letter on the screen.

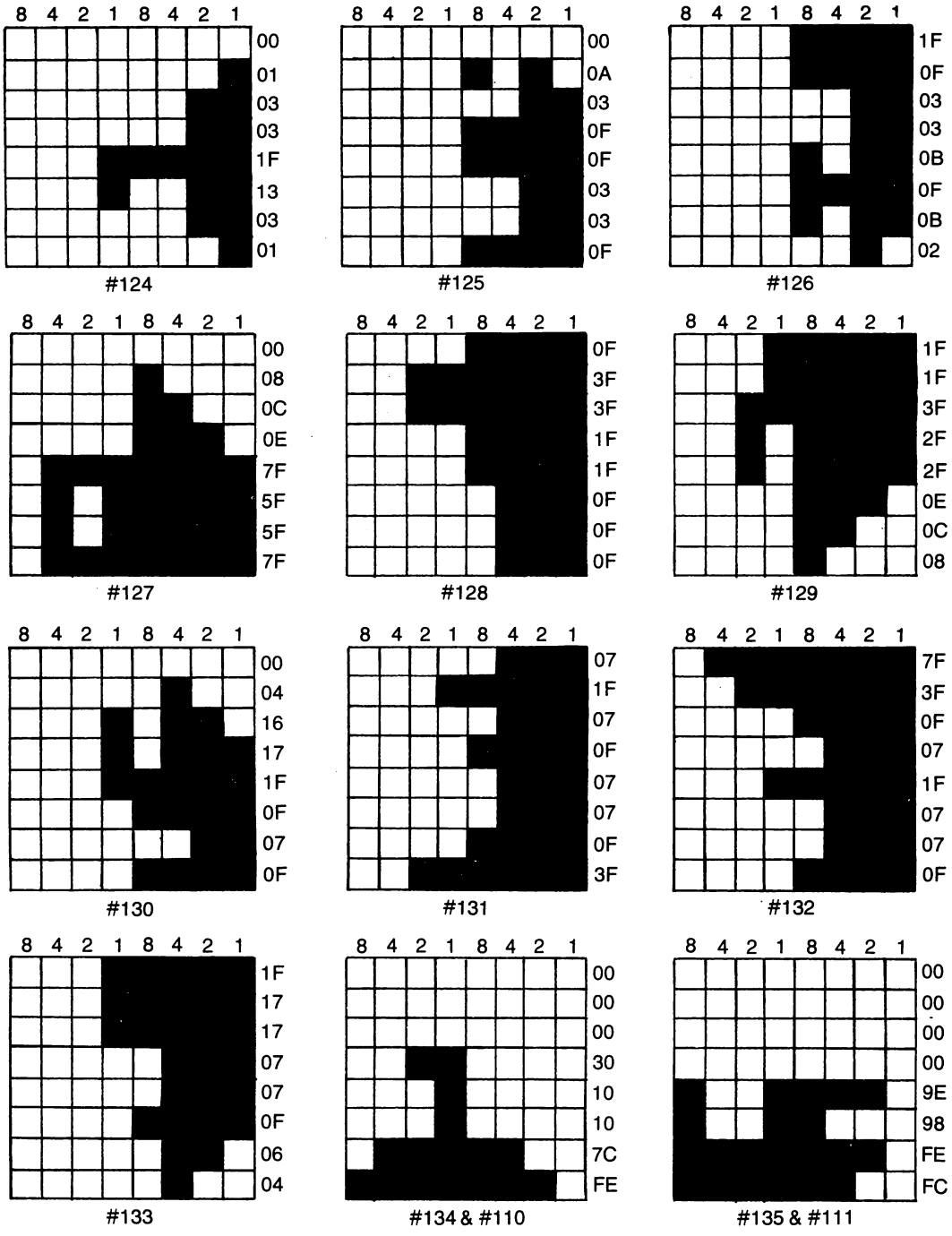
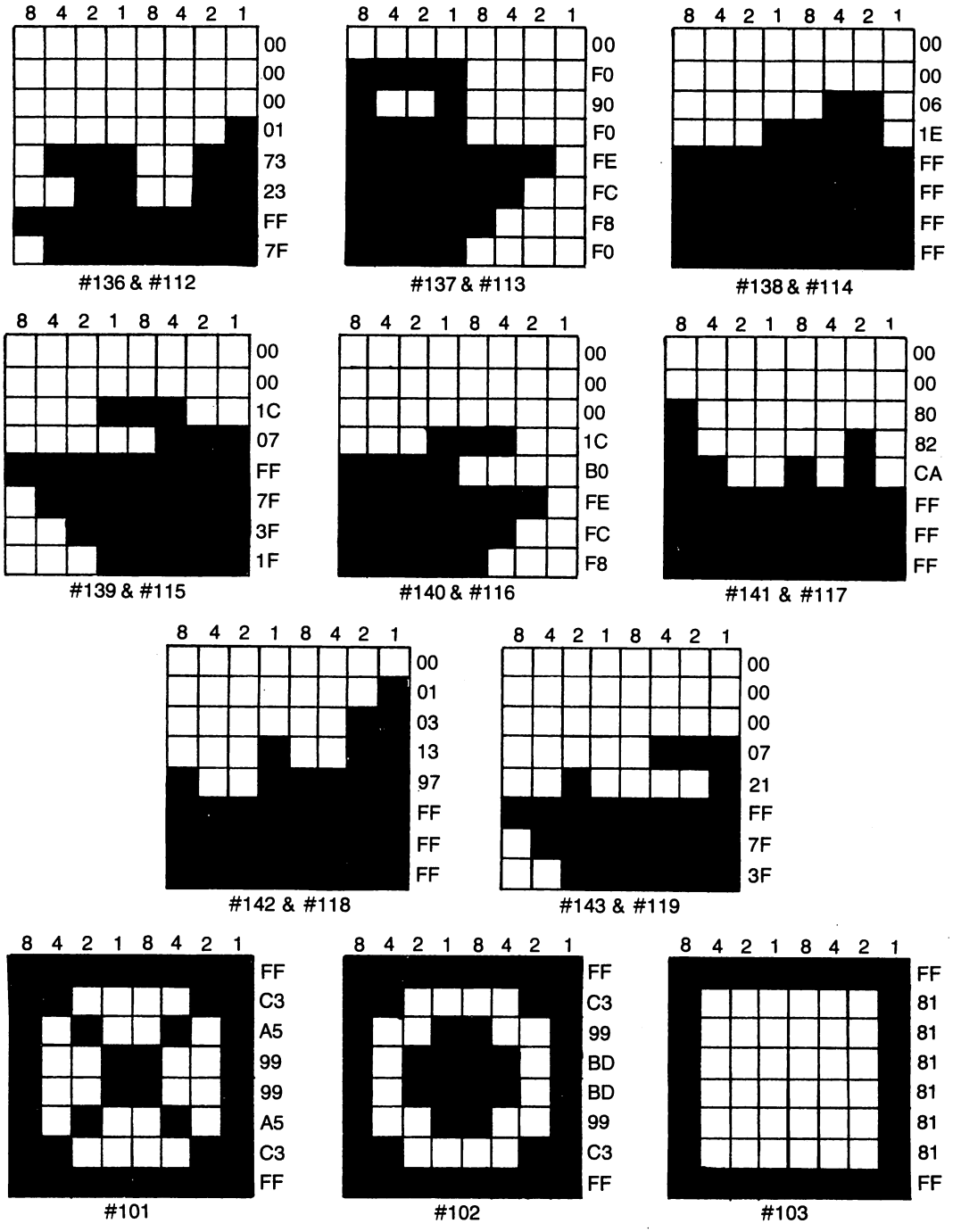


Fig. 16-11. Characters for Listing 16-6 Battleship.


8 4 2 1 8 4 2 1
00
00
00
01
73
23
FF
7F
#136 & #112
8 4 2 1 8 4 2 1
00
F0
90
F0
FE
FC
F8
F0
#137 & #113
8 4 2 1 8 4 2 1
00
00
06
1E
FF
FF
FF
FF
#138 & #114
8 4 2 1 8 4 2 1
00
00
1C
07
FF
7F
3F
1F
#139 & #115
8 4 2 1 8 4 2 1
00
00
00
1C
B0
FE
FC
F8
#140 & #116
8 4 2 1 8 4 2 1
00
00
80
82
CA
FF
FF
FF
#141 & #117
8 4 2 1 8 4 2 1
00
01
03
13
97
FF
FF
FF
#142 & #118
8 4 2 1 8 4 2 1
00
00
00
07
21
FF
7F
3F
#143 & #119
8 4 2 1 8 4 2 1
FF
C3
A5
99
99
A5
C3
FF
#101
8 4 2 1 8 4 2 1
FF
C3
99
BD
BD
99
C3
FF
#102
8 4 2 1 8 4 2 1
FF
81
81
81
81
81
81
FF
#103

```
));:: IF T1<>101 AND T1<>102 THEN SC=C-1
:: N=R :: T=T1 :: GOTO 600
560 IF R<>10 THEN T1=ASC(SEG$(H$(R+1),C,
1));:: IF T1<>101 AND T1<>102 THEN SC=C :
: N=R+1 :: T=T1 :: GOTO 600
570 IF R<>1 THEN T1=ASC(SEG$(H$(R-1),C,1
));:: IF T1<>101 AND T1<>102 THEN SC=C ::
 N=R-1 :: T=T1 :: GOTO 600
580 NEXT C :: NEXT R :: CHF=0
590 SC=INT(RND*10)+1 :: N=INT(RND*10)+1
:: T=ASC(SEG$(H$(N),SC,1));:: IF T=101 OR
 T=102 THEN 590
600 DISPLAY AT(23,13):CHR$(SC+64)&"\"&ST
R$(N)
610 IF T<>103 THEN 630 ! HIT
620 TEMP$=CHR$(102):: GOSUB 1630 :: DISP
LAY AT(23,1):" " :: GOTO 450
630 GOSUB 1610 :: TEMP$=CHR$(101):: CS=C
S+1 :: GOSUB 1630 :: CHF=1 :: DISPLAY AT
(23,1):" "
640 IF CS<20 THEN 510 ELSE 680
650 REM HUMAN GOT A HIT
660 GOSUB 1610 :: TEMP$=CHR$(101):: HS=H
S+1 :: GOSUB 1490
670 IF HS<20 THEN 460
680 IF CS=20 THEN DISPLAY AT(21,1):TAB(1
0);"I WIN !!!"
690 IF HS=20 THEN DISPLAY AT(21,1):TAB(9
);"YOU WIN !!"
700 DISPLAY AT(23,4):"PRESS 'R' TO RESTA
RT"
710 CALL KEY(0,K,S):: IF S=0 THEN 710 EL
SE IF K<>82 THEN 710
720 GOTO 160
730 REM GET LEGAL LETTER & NUMBER & DISP
LAY THEM
740 DISPLAY AT(21,1)BEEP:"ENTER A LETTER
 & NUMBER"
750 CALL KEY(0,K,S):: IF S=0 THEN 750 EL
SE IF K<65 OR K>74 THEN 750
760 SC=K-64
770 DISPLAY AT(21,25):CHR$(K)&CHR$(92)
780 N$="" :: FOR C=1 TO 3
```

continued on page 184

Line 770 prints the character string of the ASCII value of K and the slash on the screen. The character is the letter that you entered.

Line 780 clears N$. This string will store the number that you enter. The FOR . . . NEXT loop counts from one to three. You can only enter two numbers, but the computer waits for the ENTER key to be pressed. This could be the third key pressed.

Line 790 waits for a key to be pressed. When the value of S is one, the computer checks the value of K. If K is either 13 or 88, the computer will go on to line 830. If the ENTER key or X key has not been pressed, the computer continues with the next program line.

Line 800 checks the value of K to see if it is less than 49, the ASCII value for one. If the key pressed has a value less than 49, and this is the first key being pressed, then the entry is not good and the computer goes back to line 790 to wait for another key. If the value of K is greater than 57, the key pressed was not a number and the computer goes back to get another key.

Line 810 checks to see if the third key is being pressed. If it is and this key is not ENTER, the computer goes back to line 790 and waits for the ENTER key or the X key to be pressed.

Line 820 adds the character of the key pressed to the characters in N$ and prints the character on the screen. The loop continues until ENTER has been pressed.

Line 830 checks the length of N$. The computer comes to this line when the X key or the ENTER key has been pressed. If the length of N$ is zero, there have been no numbers entered and the computer goes back to line 790 to get a number. Otherwise, the value of N$ is placed in the variable N. If this value is less than one or greater than ten, the number entered is not on the grid, and the computer goes back to line 770 to erase the number on the screen. If the number is on the grid, the computer goes on to line 850.

Line 840 checks the value of K for the X key. If it is the one that was pressed, the computer goes to line 740 to get another letter and number.

Line 850 returns the computer to the main program.

Line 870 prints the question on the screen. Now the computer needs to know if the ship should be placed across or upward on the grid.

Line 880 waits for a key to be entered. The computer will loop at this line until the value of S is not zero. The value of K is then checked for 65 (A) and 85 (U). The computer will loop at this line until the A or U is pressed.

Line 890 subtracts the length of the ship from the position of the ship if the letter U was pressed. The SHP array stores the length of each ship and the SH variable is the ship that will be printed on the screen. If the ship is longer than the number of squares on the grid, the computer goes to line 1150. For example, if you were placing the battleship on the grid and wanted it to go up. You would have to place it in at least the fourth row on the grid, any row higher would be off the grid.

Line 900 checks to make sure that the ship can be placed across on the grid. If it can't, the computer goes to line 1150 to continue the program and get another entry.

Line 910 places the direction on the grid.

Line 920 checks the SH variable. If it is one, the computer does not have to check to see if this ship will run into any other ships since it is the only ship on the grid!

Line 930 checks the value of K to see if an A was pressed. If an A was not pressed, the computer is directed to line 960.

Lines 940-950 check the squares across to see if this ship will run into any other ships. If it does, the computer is sent to line 1150. If the ship can be placed on the grid, the computer goes to line 980.

Lines 960-970 check the squares up the grid to see if the ship will run into any other ships. If it does, the computer is sent to line 1150. If the ship can be placed on the grid, the computer continues with the next program line.

Line 980 places the horizontal ship in the temporary string when the letter A is pressed. If the U is pressed, the vertical ship is placed in the temporary string.

Line 990 finds the length of the ship. If the ship is to

```
790 CALL KEY(0,K,S)!! IF S=0 OR S=-1 THE
N 790 ELSE IF K=13 OR K=88 THEN 830
800 IF K<49 AND C=1 THEN 790 ELSE IF K>5
7 THEN 790
810 IF C=3 AND K<>13 THEN 790
820 N$=N$&CHR$(K)!! DISPLAY AT(21,26+C)!
CHR$(K)!! NEXT C
830 IF LEN(N$)=0 THEN 790 ELSE IF K=13 T
HEN N=VAL(N$)!! IF N<1 OR N>10 THEN 770
ELSE 850
840 IF K=88 THEN 740
850 RETURN
860 REM GET UP OR ACROSS INFORMATION AND
 DISPLAY IT
870 DISPLAY AT(23,5)BEEP!"UP OR ACROSS ?
"
880 CALL KEY(0,K,S)!! IF S=0 THEN 880 EL
SE IF K<>65 AND K<>85 THEN 880
890 IF K=85 THEN IF (N-SHP(SH))<0 THEN 1
150
900 IF K=65 THEN IF ((11-SC)-SHP(SH))<0
THEN 1150
910 IF K=65 THEN DISPLAY AT(23,5)!"
 ACROSS" ELSE DISPLAY AT(23,5)!"UP"
920 IF SH=1 THEN 980 ! DON'T NEED TO CHE
CK FOR OTHER SHIPS
930 IF K<>65 THEN 960
940 FOR COL=SC TO SC+(SHP(SH)-1)!! IF AS
C(SEG$(H$(N),COL,1))<>103 THEN 1150
950 NEXT COL !! GOTO 980
960 FOR ROW=N TO N-(SHP(SH)-1)STEP -1 !!
 IF ASC(SEG$(H$(ROW),SC,1))<>103 THEN 11
50
970 NEXT ROW
980 IF K=65 THEN TEMP$=SH$(SH)ELSE TEMP$
=SV$(SH)
990 LN=LEN(TEMP$)!! IF K=85 THEN 1010
1000 DISPLAY AT(7+N,9+SC)SIZE(LN)!TEMP$
!! GOTO 1030
1010 FOR C=1 TO LEN(TEMP$)!! DISPLAY AT(
8+N-C,9+SC)SIZE(1)!SEG$(TEMP$,C,1)!! NEX
T C
1020 REM NOW WAIT FOR AN 'ENTER' OR 'X'
(DELETE)
```

continued on page 186

go up on the grid, the computer goes on to program line 1010.

Line 1000 places the ship across on the grid at the correct location and the program continues at line 1030.

Line 1010 places the ship up on the grid at the correct location.

Line 1030 waits for a key to be pressed. Only two keys will be accepted—the ENTER key or the X key. If the ship is in the position that you would like, press ENTER and the program will continue. If you would like to erase the ship, press the X key, the ship will be erased, and you will be able to try it at a different position. The computer will loop at this line until ENTER or the X key has been pressed.

Line 1040 checks the KY variable for the X value. If the X key was pressed, the computer will be sent to line 1110 to erase the ship.

Lines 1050-1090 check the value of SH to see if any ship should be erased from the screen. If SH equals any of these values, the ship that was placed on the grid will be erased from the row of ships under the grid, and the next ship will change colors.

Line 1100 checks the direction that the ship has been placed in on the screen and directs the computer to the correct routine to place the ship in the other array as well.

Line 1110 erases a ship that was placed upward on the grid.

Line 1120 erases a ship that was placed across on the grid.

Line 1130 sends the computer to line 1150 to set the flag.

Line 1150 sets the FLAG variable to one. When this flag is set, the computer knows that the last ship was erased, and it should not go on to the next ship. The computer returns to the main program.

Lines 1170-1200 puts the ship in the player's array vertically. One section of this ship is placed row by row into the array.

Lines 1220-1240 place the ship in the player's array. This time the ship is placed in the array horizontally. The elements of the array before and after the ships location are placed into temporary strings, then concatenated into the array.

Line 1260 is the flag that is set when the computer makes a bad choice.

Line 1280 checks the computer's choice to place a ship upward on the grid. If the ship cannot fit on the grid, the computer is directed to line 1260 to make another choice.

Line 1290 checks to see if the ship can fit across on the grid. If it cannot the computer will go to line 1260 and return to the main program.

Line 1300 checks to see if this is the first ship. If it is, then there is no need to check the grid for other ships.

Line 1310 sends the computer to line 1340 to check the grid going up.

Lines 1320-1330 see if the ship can be placed across on the grid. If it can, the computer is directed to line 1360 to place the ship on the grid. If it cannot, the computer must make another choice.

Lines 1340-1350 check to see if the ship can be placed upward on the grid. If it can, the computer continues with the program. If it cannot, the computer must make another choice.

Line 1360 places the horizontal ship into the temporary string, if the ship should be placed across on the grid. It places the vertical ship in the array if the ship should be placed upward on the grid.

Line 1370 finds the length of the ship and sends the computer to line 1440 if the ship will be positioned horizontally.

Lines 1390-1420 position the ship vertically in the computer's array. Each part of the ship must be placed in a different row of the grid.

Lines 1440-1460 position the ship horizontally in one row of the grid.

Line 1470 places the screen grid on the screen.

Lines 1490-1570 place the hit or miss marker in the computer's grid and on the grid on the screen. First the marker that is stored in the temporary string is placed on the screen. If the player's score is more than zero, it is printed on the screen. Then the computer checks the value of the marker. If it is 102, it is a miss and the computer is sent to the subroutine that makes the miss sound. The marker is placed in the computer's string array in the correct position. It is then placed into

```
1030 CALL KEY(0,KY,S):: IF S=0 THEN 1030
 ELSE IF KY<>13 AND KY<>88 THEN 1030
1040 IF KY=88 THEN 1110
1050 IF SH=1 THEN DISPLAY AT(19,8)SIZE(8
):"      "&CH$
1060 IF SH=3 THEN DISPLAY AT(19,13)SIZE(
6):"     "&DH$
1070 IF SH=6 THEN DISPLAY AT(19,17)SIZE(
4):"    "&SBH$
1080 IF SH=10 THEN DISPLAY AT(19,20):" "
1090 DISPLAY AT(23,1):" "
1100 IF K=65 THEN 1220 ELSE 1170
1110 IF K=85 THEN CALL VCHAR(N-LEN(TEMP$
)+8,SC+11,103,LEN(TEMP$))
1120 IF K=65 THEN CALL HCHAR(N+7,SC+11,1
03,LEN(TEMP$))
1130 REM BAD INPUT OR DELETE
1140 REM ERASE BOTTOM LINE & SET FLAG
1150 FLAG=1 :: DISPLAY AT(23,5):" " :: R
ETURN
1160 REM PUT VERTICAL SHIP IN HUMAN'S AR
RAY
1170 FOR C=1 TO LEN(TEMP$)
1180 IF SC>1 THEN TF$=SEG$(H$(N-C+1),1,S
C-1)ELSE TF$=""
1190 IF SC<10 THEN TL$=SEG$(H$(N-C+1),SC
+1,10-SC)ELSE TL$=""
1200 H$(N-C+1)=TF$&SEG$(TEMP$,C,1)&TL$ :
: NEXT C :: RETURN
1210 REM PUT HORIZONTAL SHIP INTO HUMAN'
S ARRAY
1220 IF SC>1 THEN TF$=SEG$(H$(N),1,SC-1)
ELSE TF$=""
1230 IF SC<10 THEN TL$=SEG$(H$(N),SC+LN,
10-(SC+LN-1))ELSE TL$=""
1240 H$(N)=TF$&TEMP$&TL$ :: RETURN
1250 REM BAD CHOICE - COMPUTER
1260 FLAG=1 :: RETURN
1270 REM CHECK UP & ACROSS COORDINATES F
OR COMPUTER
1280 IF K=85 THEN IF (N-SHP(SH))<0 THEN
1260
```

continued on page 188

the string array that places the characters on the screen.

Line 1590 is the delay loop. This leaves the information on the screen for a few seconds before the next part of the program.

Line 1610 is the flash routine. When a hit is made, the computer is sent to this subroutine to make the crackling sound and change the screen colors.

Lines 1630-1670 place the computer's move in the player's grid. If the computer's score is greater than zero, it is placed on the screen. If the marker was a miss, the computer is sent to the subroutine in line 1690 to make the miss sound. The marker is placed in the player's array, and the computer is sent to line 1590 before returning to the main program. This routine is a subroutine. By sending the computer to line 1590 with a GOTO statement, the computer will return to the correct line in the program.

Line 1690 is the sound routine for the miss.

USING JOYSTICKS

Built into your TI-99/4A BASIC are commands for reading the positions of joysticks. The keyboard is a good way to enter information, but a joystick can make a program easier to use. If you have ever played an arcade type game that used the keyboard to move characters, you will understand why the joystick is a better choice. The user has no chance to press the wrong key and then wonder why the character isn't moving in the direction expected. A joystick can be used in programs other than arcade games.

CALL JOYST

This command is used to find out which direction the joystick has been moved. The format for this command is:

CALL JOYSTICK(unit,x-coordinate,y-coordinate)

The unit number indicates which joystick is being read. Your TI-99/4A can use two joysticks in a program. Use a one for the first joystick and a two for the second joystick. The next variable is the x-coordinate. If this value is a zero, the joystick has not moved to the left or right. If the variable contains a four, the joystick was moved to the right. If it is a negative four, the joystick was moved to the left. The y-coordinate determines whether the joystick has moved up or down. If the Y variable is a four, the joystick has been moved up, a negative four means the joystick has moved down. If the values of both variables are zero, the joystick has not moved.

In addition to the JOYST command, the CALL KEY command can be used. The JOYST command only checks which direction the joystick is pointing to. It does not check to see if the fire button has been pressed. To check for the fire button, use:

CALL KEY (1,key,status)

The one indicates the first joystick or the keys on the left side of the keyboard. If you wanted to check the second joystick, you would use a two. The key variable is the ASCII value of the key that has been pressed or the value of the fire button. When the fire button has been pressed, the variable will be an 18. The status will be set to one when the fire button or a key has been pressed. The program in Listing 16-7 (flowcharted in Fig. 16-12) demonstrates using a joystick with a menu.

Listing 16-7

Line 130 places a character pattern in C$. This new character will be used as our pointer (Fig. 16-13). The character will be the 143 number in the character set. Its color is set to dark blue and the background to transparent. The character is moved into P$. Whenever we want to print the character on the screen, we can just print P$.

Line 140 places a message on the screen. The ALPHA LOCK key must be up in order for this program to work. If the key is down, the computer will not read the joystick correctly.

Lines 150-160 clear the screen and place the menu on it. This is an example of a menu that might be used in many programs.

Line 170 sets the PR variable to 4. This variable will indicate which row the pointer should be printed in.

Line 180 places the pointer (P$) on the screen. It is placed in the seventh column in the row set by

```
1290 IF K=65 THEN IF ((11-SC)-SHP(SH))<0
 THEN 1260
1300 IF SH=1 THEN 1360 ! DON'T NEED TO C
HECK FOR OTHER SHIPS
1310 IF K<>65 THEN 1340
1320 FOR COL=SC TO SC+(SHP(SH)-1):: IF A
SC(SEG$(COM$(N),COL,1))<>103 THEN 1260
1330 NEXT COL :: GOTO 1360
1340 FOR ROW=N TO N-(SHP(SH)-1)STEP -1 :
: IF ASC(SEG$(COM$(ROW),SC,1))<>103 THEN
 1260
1350 NEXT ROW
1360 IF K=65 THEN TEMP$=SH$(SH)ELSE TEMP
$=SV$(SH)
1370 LN=LEN(TEMP$):: IF K=65 THEN 1440
1380 REM PUT VERTICAL SHIP IN COMPUTER'S
 ARRAY
1390 FOR C=1 TO LEN(TEMP$)
1400 IF SC>1 THEN TF$=SEG$(COM$(N-C+1),1
,SC-1)ELSE TF$=""
1410 IF SC<10 THEN TL$=SEG$(COM$(N-C+1),
SC+1,10-SC)ELSE TL$=""
1420 COM$(N-C+1)=TF$&SEG$(TEMP$,C,1)&TL$
 :: NEXT C :: RETURN
1430 REM PUT HORIZONTAL SHIP INTO COMPUT
ER'S ARRAY
1440 IF SC>1 THEN TF$=SEG$(COM$(N),1,SC-
1)ELSE TF$=""
1450 IF SC<10 THEN TL$=SEG$(COM$(N),SC+L
N,10-(SC+LN-1))ELSE TL$=""
1460 COM$(N)=TF$&TEMP$&TL$ :: RETURN
1470 FOR R=8 TO 17 :: DISPLAY AT(R,9+(R=
17)):STR$(R-7)&SCR$(R-7):: NEXT R :: RET
URN
1480 REM PLACE VALUE IN TEMP$ ON COMPUTE
R'S GRID
1490 DISPLAY AT(7+N,9+SC)SIZE(1):TEMP$ :
: IF HS>0 THEN DISPLAY AT(3,25):HS
1500 IF ASC(TEMP$)=102 THEN GOSUB 1690
1510 IF SC>1 THEN TF$=SEG$(COM$(N),1,SC-
1)ELSE TF$=""
1520 IF SC<10 THEN TL$=SEG$(COM$(N),SC+1
,10-SC)ELSE TL$=""
```

continued on page 189

```
1530 COM$(N)=TF$&TEMP$&TL$
1540 REM PLACE VALUE IN TEMP$ ON SCREEN'
S GRID
1550 IF SC>1 THEN TF$=SEG$(SCR$(N),1,SC-
1)ELSE TF$=""
1560 IF SC<10 THEN TL$=SEG$(SCR$(N),SC+1
,10-SC)ELSE TL$=""
1570 SCR$(N)=TF$&TEMP$&TL$ :: T=1000
1580 REM DELAY ROUTINE
1590 FOR DELAY=1 TO T :: NEXT DELAY :: R
ETURN
1600 REM FLASH SCREEN & SOUND
1610 FOR F=1 TO 3 :: CALL SCREEN(16):: C
ALL SOUND(100,110,5,110,5,110,5,-8,0)::
CALL SCREEN(2):: NEXT F :: RETURN
1620 REM PUT SHOT IN TEMP$ ON HUMAN'S GR
ID
1630 DISPLAY AT(7+N,9+SC)SIZE(1):TEMP$ :
: IF CS>0 THEN DISPLAY AT(3,3)SIZE(4):CS
1640 IF ASC(TEMP$)=102 THEN GOSUB 1690
1650 IF SC>1 THEN TF$=SEG$(H$(N),1,SC-1)
ELSE TF$=""
1660 IF SC<10 THEN TL$=SEG$(H$(N),SC+1,1
0-SC)ELSE TL$=""
1670 H$(N)=TF$&TEMP$&TL$ :: T=1000 :: GO
TO 1590
1680 REM MISS SOUND
1690 FOR S=3 TO 1 STEP -1 :: CALL SOUND(
50,110,30,110,30,S*1000,0,-4,0):: NEXT S
 :: RETURN
```

PR. The delay loop holds the computer at this line for a few seconds. Without this delay, it would be very difficult to place the pointer at the correct position.

Line 190 uses the CALL KEY command to see if the fire button on the joystick has been pressed. If the S variable is one then the fire button or a key has been pressed. If the K variable is 18, then the fire button has been pressed and the computer will be directed to line 270 to continue with the program.

Line 200 uses the JOYST command to see if the joystick has been moved. In this program, we are only interested in moving the pointer up or down. The X variable will change if the joystick has been moved left or right. If it has, the computer will be sent back to line 190. If X is zero, the computer will continue with the next program line.

Line 210 checks the Y variable. This variable will

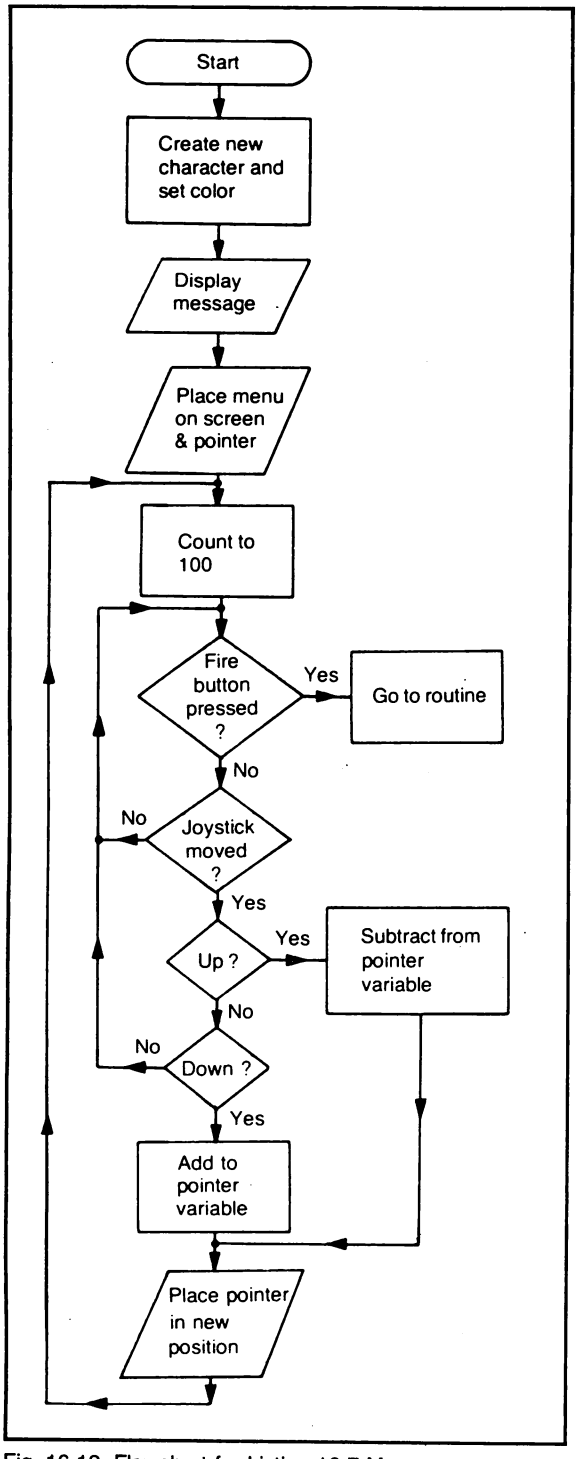


Fig. 16-12. Flowchart for Listing 16-7 Menu.

change if the joystick has been moved up or down. If the variable is a zero, then the joystick has not been moved and the computer is directed to line 190 to check for another input.

Line 220 checks the value of Y for a four, which indicates the joystick has been moved up. The computer will be sent to line 250 to move the pointer up one item on the menu.

Line 230 moves the pointer down one item on the menu. The computer will use this line if the value of Y is negative four. We do not have to check for the value of Y since the last two program lines checked for the other two values that Y could be. The computer erases the pointer from its present position. Four is added to the value of PR because the items on the menu are four rows apart. When you move down on the screen, the row numbers increase. The fifth item on the menu is at row 20. PR is checked to see if it has passed row 20. If it has, the variable is reset to 20.

Line 260 sends the computer back to line 180 to print the pointer at the new position on the screen.

Line 250 erases the pointer from the screen and subtracts four from PR. When we move the pointer up the screen, the row values decrease. The first item on the menu is printed at row four. The value of PR is checked to see if the pointer would be above the first item. If it is less than four, the variable is reset to four.

Line 260 sends the computer back to line 280 to print the pointer on the screen again.

Line 270 begins what would be the main program.

Line 280 ends the program.

Any standard joystick can be used with your TI-99/4A. A special Y-shaped cable will allow you to attach two joysticks to the port on the left side of your keyboard.

USING THE SPEECH SYNTHESIZER

It is possible for your TI-99/4A to talk to you through a speech synthesizer. Texas Instruments manufactures a speech synthesizer that is compatible with your computer. The EXTENDED BASIC cartridge has a built-in set of command and vocabul-

Listing 16-7

```
100 REM LISTING 16-7
110 REM MENU
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 C$="003F787878700000" :: CALL CHAR(1
43,C$):: CALL COLOR(14,5,1):: P$=CHR$(14
3)
140 DISPLAY AT(12,1)ERASE ALL:"'ALPHA LO
CK' KEY MUST BE UP" :: FOR DELAY=1 TO 15
00 :: NEXT DELAY
150 DISPLAY AT(4,9)ERASE ALL:"1. NUMBERS
" :: DISPLAY AT(8,9):"2. LETTERS" :: DIS
PLAY AT(12,9):"3. COLORS"
160 DISPLAY AT(16,9):"4. SHAPES" :: DISP
LAY AT(20,9):"5. SIZES"
170 PR=4 ! ROW OF THE POINTER
180 DISPLAY AT(PR,7)SIZE(1):P$ :: FOR DE
LAY=1 TO 100 :: NEXT DELAY !PAUSE BETWEE
N MOVES
190 CALL KEY(1,K,S):: IF S=1 AND K=18 TH
EN 270 ! SELECTION HAS BEEN MADE
200 CALL JOYST(1,X,Y):: IF X<>0 THEN 190
210 IF Y=0 THEN 190
220 IF Y=4 THEN 250
230 DISPLAY AT(PR,7)SIZE(1):" " :: PR=PR
+4 :: IF PR>20 THEN PR=20
240 GOTO 180
250 DISPLAY AT(PR,7)SIZE(1):" " :: PR=PR
-4 :: IF PR<4 THEN PR=4
260 GOTO 180
270 REM GO NOW TO THE UNIT SELECTED
280 END
```

ary words that can be used with this speech synthesizer.

Although it is possible to use other commercially available speech synthesizers with your TI-99/4A, we will only explain the commands that are used with the Texas Instruments' one. Consult your manufacturer's booklet for the other speech synthesizers.

SAY

With this command you can have the speech synthesizer say any word that is in its vocabulary. The entire word list is in the EXTENDED BASIC book. If you ask the computer to say a word that is not in its vocabulary, it will spell the word instead of saying it. The format for this command is:

CALL SAY("HELLO")

More than one word may be included between the quotation marks. Single letters and numbers can also be spoken. Resident phrases must be enclosed in pound and quotation marks.

CALL SAY (" TEXAS INSTRUMENTS "0

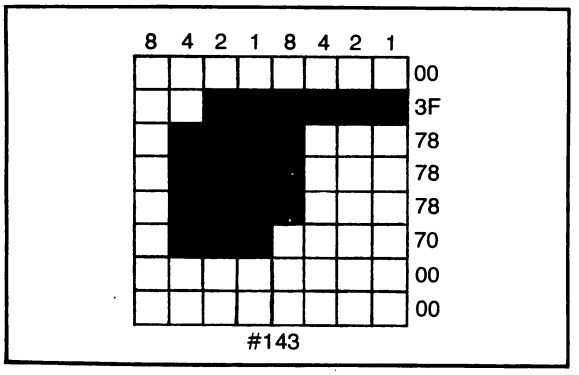


Fig. 16-13. Character for Listing 16-7 Menu.

CALL SPGET

This command is similar to the CALL CHARPAT command. Every word that the TI-99/4A says follows a particular pattern. This pattern can be altered or changed to create new words that are not in the computer's present vocabulary. The format for this command is:

CALL SPGET("word",string variable)

The word in quotation marks is a word from the computer's vocabulary. The string variable is any string variable that you want to use. The computer will place in the string variable the codes that make the sound for the word. This string can be very long since it takes several codes for the computer to make a sound.

If you would like the computer to say the word that is in the string, use the SAY command with a comma before the string variable.

CALL SAY(,R$)

The codes tell the computer to make sounds called *phonemes*. Phonemes are the sounds that make up the letter sounds to form words. For example, there are six vowels in the English language (counting Y), but they can make over 26 different and distinct sounds. The phoneme code tells the computer which sound to make. These phonemes can be changed or added together to make new words or sounds. The program in Listing 16-8C allows you to take two words and add the phonemes from one word to the phonemes from the other word to create a new word.

In the program in Listing 16-8C (flowcharted in Fig. 16-14) we will shorten the phonemes of words and add them to the phonemes of another word to make the computer say a word that is not in its resident vocabulary. The program can store up to 40 words. These words can be stored on cassette and used in other programs. The second Listing, 16-8D, is the same, except that the words are stored on disk instead of cassette.

Listing 16-8C

Line 130 traps errors. If an error has been entered the computer will continue without displaying the message.

Line 140 eliminates the zero element of the array. SPEAK$ array will store the words and their phonemes. The C variable is set to one. This variable will count the number of words entered.

Line 150 asks you to enter a word. The word will be stored in WORD$. If you do not enter a word, the computer will remain at this line until you do.

Line 160 uses the SAY command to say the word. If the word is in the computer's resident vocabulary, the computer will say the word. If it is not, the computer will spell the word.

Line 170 sends the computer to the subroutine at line 430. This subroutine is used throughout the program to ask you if you like the sound of the word or to repeat it. If you enter an "N," the computer will go to line 150 and you can enter another word. If you enter an "A" to say the word again, the computer will go to line 160 to say the word. If you enter a "Y," the computer will continue with the next program line.

Line 180 asks you to enter another word. If you do not have another word to enter, you can just press ENTER and the computer will continue at program line 210.

Line 190 says the word that you entered.

Line 200 uses the subroutine at line 430 to find out if you liked the way the word sounded. If you enter an "N," you can enter another word. Enter an "A" to hear the word again.

Line 210 erases the screen and prints one or both words near the top of the screen.

Line 220 uses the SPGET command and places the phonemes for the first word into WO1$. If a second word was entered, the phonemes for this word will be stored in WO2$.

Line 230 finds the length of the two phoneme strings. This length will be used to determine how many bytes can be used and/or removed from the string when we try to change the sound of the word.

Line 240 tells you how many bytes there are in the first phoneme string. We subtract three from the length of the string because the first three bytes of the string cannot be removed. You are then asked how many bytes you want to use.

Line 250 accepts the number that you enter. If you try to use more bytes than are in the string, the line will repeat itself.

Line 260 places the first two bytes of the phoneme string concatenated with the number of bytes that you want to use in W$. We leave the first two bytes of the original string intact. The third byte is changed to the number of bytes that will follow it. If this number is not correct, an error will occur when the computer tries to say the word. The first part of the phoneme string, beginning with the fourth byte, is added to these three bytes.We will only be adding sounds from the beginning of the string.

Line 270 says the word in W$. The W$ contains the phonemes or codes for the word, so we must precede it with a comma. The comma tells the computer that the first element, a word in a string or within quotation marks, has not been entered, so go on to the second element of the command, the string with the phonemes.

Line 280 asks you if you liked the way the word If you did not, enter an "N." The computer will remove that phoneme string from W$ and ask you to enter another length. You can keep trying different lengths until the part of the word that you want the computer to say sounds right to you. If you enter an "A," the computer will say the word again. If you liked the way the word sounded enter a "Y."

Line 290 checks to see if a second word was entered. If the string is empty, the computer will go on to line 350.

Line 300 shows you the length of the second string. Again, three is subtracted from its length.

Line 310 accepts the number for the number of bytes that you want to use. Again, we will be removing the first sounds of the word.

Line 320 places the first two bytes, along with the number of bytes following them and their contents, in W2$.

Line 330 uses the SAY command to say the word made up of the phonemes in W2$.

Line 340 uses the subroutine at line 430 to ask you if you liked the way the word sounded. Enter an "N" to try a different number of bytes, an "A" to hear the word again, and a "Y" to continue the program.

Line 350 says the two phoneme strings together.

Line 360 asks you if you liked the way it sounded. Respond as you did before.

Line 370 finds the length of just the first phoneme string, or the length of both phoneme strings together. This length will be used when we are saving the program to cassette.

Line 380 places the phoneme string in the second element of SPEAK$ array. The first element of the array will be used for the word that is spoken.

Line 390 asks you to enter the word for the phoneme string that you created.

Line 400 concatenates the word with the phoneme string. A slash mark is placed between the actual word and its phonemes. This will help us identify where one word stops and its phoneme string begins. If the length of both strings together is longer than 192, a message saying so will be printed on the screen, and the computer will return to line 150. The string cannot be longer than 192 bytes. If it is, the computer will not be able to save it onto the cassette.

Line 410 asks you if you have any more words to enter into the array. If no letter was entered, the computer will loop at this line until one is.

Line 420 adds one to the count of C. If the "N" was entered, or the count is more than 40, the computer will be directed to line 470 to save the

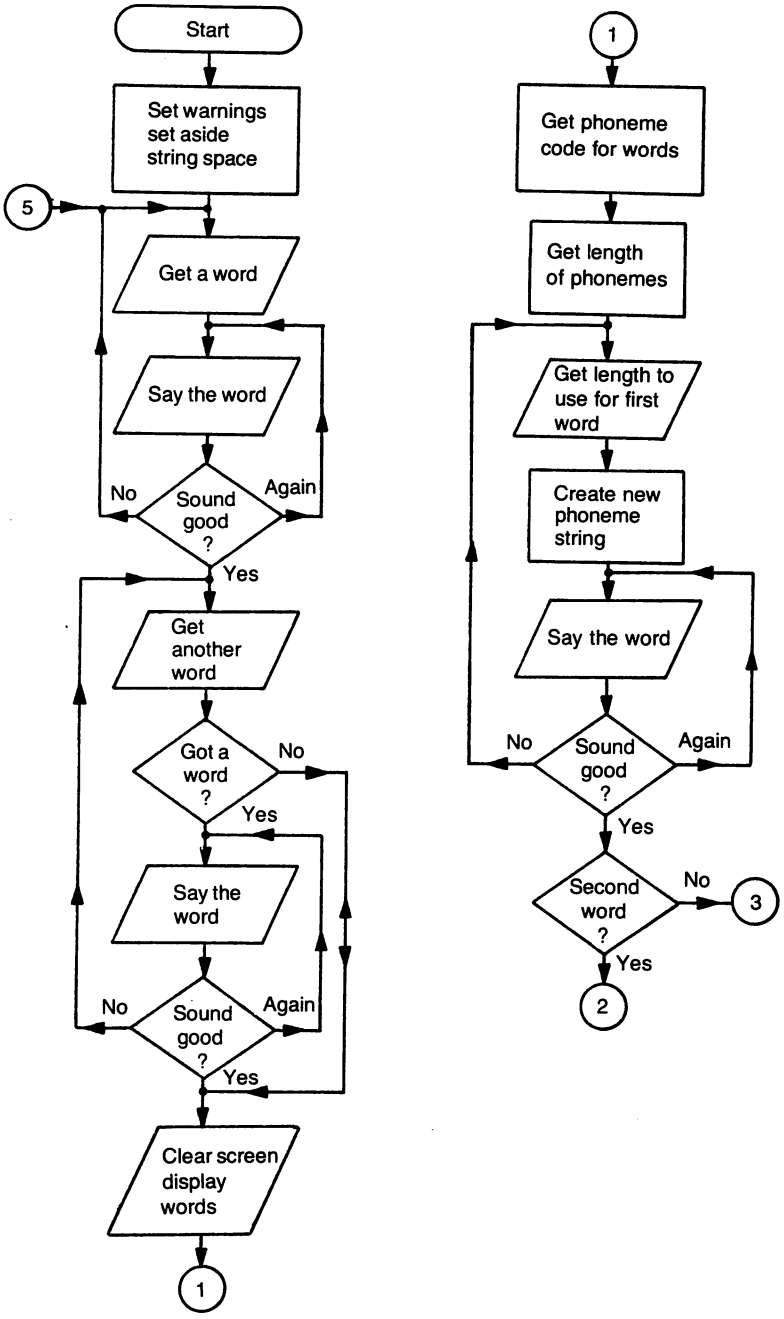
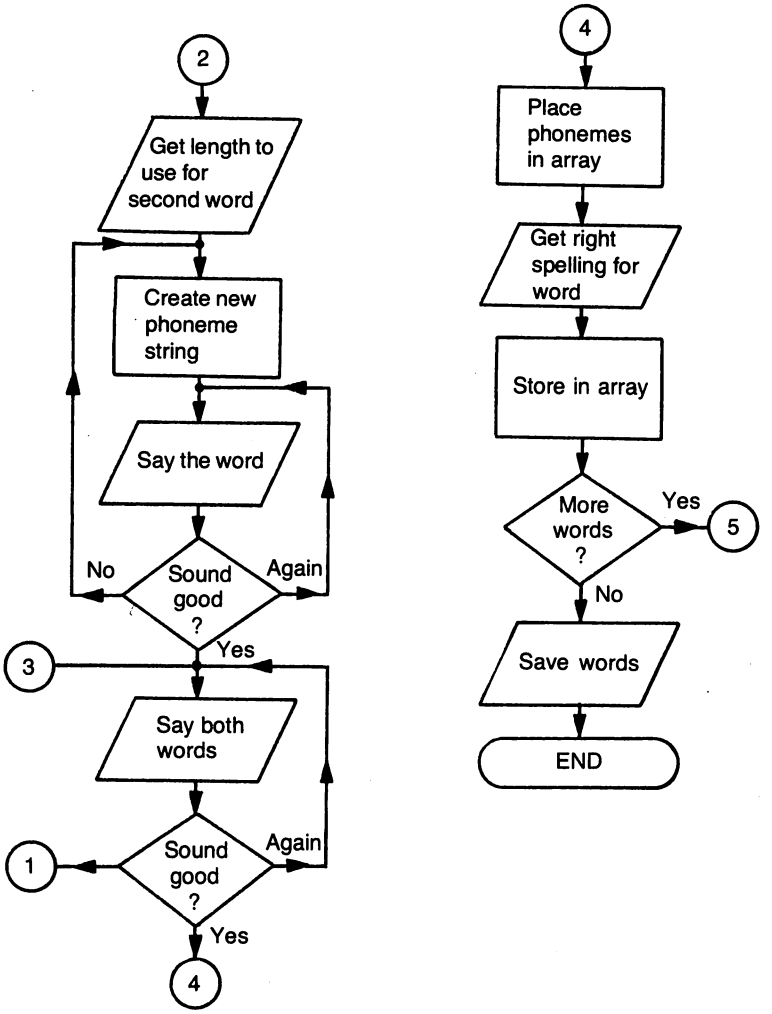


Fig. 16-14. Flowchart for Listing 16-8 C&D Making Words.



Listing 16-8C

```
100 REM LISTING 16-8C
110 REM MAKING WORDS - CASSETTE
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 ON WARNING NEXT .
140 OPTION BASE 1 :: DIM SPEAK$(40,2)::
C=1
150 DISPLAY AT(2,2)ERASE ALL:"ENTER A WO
RD" :: ACCEPT AT(4,4)BEEP:WORD$ :: IF WO
RD$="" THEN 150
```

```
160 CALL SAY(WORD$)
170 GOSUB 430 :: IF V$="N" THEN 150 ELSE
 IF V$="A" THEN 160
180 DISPLAY AT(6,2):"ENTER SECOND WORD"
:: ACCEPT AT(8,4)BEEP:WORD2$ :: IF WORD2
$="" THEN 210
190 CALL SAY(WORD2$)
200 GOSUB 430 :: IF V$="N" THEN 180 ELSE
 IF V$="A" THEN 190
210 DISPLAY AT(2,2)ERASE ALL:WORD$,WORD2
$
220 CALL SPGET(WORD$,W01$):: IF WORD2$<>
"" THEN CALL SPGET(WORD2$,W02$)
230 W=LEN(W01$):: IF WORD2$<>"" THEN W2=
LEN(W02$)
240 DISPLAY AT(4,2):"THE FIRST WORD IS";
W-3:"BYTES LONG":"USE HOW MANY?"
250 ACCEPT AT(7,13)VALIDATE(DIGIT):B ::
IF B>W-3 THEN 250
260 W$=SEG$(W01$,1,2)&CHR$(B)&SEG$(W01$,
4,B)
270 CALL SAY(,W$)
280 GOSUB 430 :: IF V$="N" THEN W$="" ::
 GOTO 250 ELSE IF V$="A" THEN 270
290 IF WORD2$="" THEN 350
300 DISPLAY AT(9,2):"THE SECOND WORD IS"
;W2-3:"BYTES LONG":"USE HOW MANY?"
310 ACCEPT AT(12,13)VALIDATE(DIGIT):B2 :
: IF B2>W2-3 THEN 310
320 W2$=SEG$(W02$,1,2)&CHR$(B2)&SEG$(W02
$,4,B2)
330 CALL SAY(,W2$)
340 GOSUB 430 :: IF V$="N" THEN W2$="" :
: GOTO 300 ELSE IF V$="A" THEN 330
350 IF WORD2$="" THEN CALL SAY(,W$)ELSE
CALL SAY(,W$,,W2$)
360 GOSUB 430 :: IF V$="N" THEN 210 ELSE
 IF V$="A" THEN 350
370 IF WORD2$="" THEN L=LEN(W$)ELSE L=LE
N(W$&W2$)
380 IF WORD2$="" THEN SPEAK$(C,2)=CHR$(L
)&W$ ELSE SPEAK$(C,2)=CHR$(L)&W$&W2$
390 DISPLAY AT(14,1):"ENTER PROPER NAME
```

```
FOR THIS":"WORD" :: ACCEPT AT(15,6)BEEP:
NAME$ :: IF NAME$="" THEN 390 ELSE SPEAK
$(C,1)=NAME$
400 L=MAX(LEN(SPEAK$(C,1)),LEN(SPEAK$(C,
2))):: IF L>254 THEN DISPLAY AT(14,1)BEE
P:"TOO LONG TO SAVE !" :: GOSUB 450 :: G
OTO 150
410 DISPLAY AT(14,1):"ANY MORE (Y/N) ?"
:: DISPLAY AT(15,1):"" :: ACCEPT AT(14,1
8)BEEP VALIDATE("YN")SIZE(1):V$ :: IF V$
="" THEN 410
420 C=C+1 :: IF V$="N" OR C>40 THEN 470
ELSE 150
430 DISPLAY AT(14,2):"SOUND OK (Y/N/A) ?
" :: ACCEPT AT(14,21)VALIDATE("ANY")SIZE
(1):V$ :: IF V$="" THEN 430
440 DISPLAY AT(14,2):"" :: RETURN
450 FOR DELAY=1 TO 2000 :: NEXT DELAY ::
 RETURN
460 REM SAVE SPEECH WORDS & THEIR SOUNDS
470 DISPLAY AT(12,1)ERASE ALL:"ENTER NAM
E FOR FILE" :: ACCEPT AT(14,19)BEEP SIZE
(10):FILENAME$ :: IF FILENAME$="" THEN 4
70
480 DISPLAY AT(12,1)ERASE ALL:"SAVING TO
 DISK - PLEASE WAIT"
490 OPEN #1:"DSK1."&FILENAME$,SEQUENTIAL
,DISPLAY ,UPDATE,VARIABLE 254
500 PRINT #1:C-1
510 FOR CT=1 TO C-1
520 PRINT #1:SPEAK$(CT,1)
530 PRINT #1:SPEAK$(CT,2)
540 NEXT CT
550 CLOSE #1
560 DISPLAY AT(14,9)ERASE ALL BEEP:"ALL
DONE !"
570 END
```

words; otherwise it will go to line 150 to get another word.

Line 430 is the subroutine that asks you if the word sounds OK. Only the letters "A," "N," and "Y" will be accepted. If the ENTER key is pressed without entering a letter, the computer will loop at this line.

Line 440 erases the message from the screen and the program returns to the main program.

Line 470 begins the subroutine that saves the words

Listing 16-8D

```
100 REM LISTING 16-8D
110 REM MAKING WORDS - DISK
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 ON WARNING NEXT
140 OPTION BASE 1 :: DIM SPEAK$(40,2)::
C=1
150 DISPLAY AT(2,2)ERASE ALL:"ENTER A WO
RD" :: ACCEPT AT(4,4)BEEP:WORD$ :: IF WO
RD$="" THEN 150
160 CALL SAY(WORD$)
170 GOSUB 430 :: IF V$="N" THEN 150 ELSE
 IF V$="A" THEN 160
180 DISPLAY AT(6,2):"ENTER SECOND WORD"
:: ACCEPT AT(8,4)BEEP:WORD2$ :: IF WORD2
$="" THEN 210
190 CALL SAY(WORD2$)
200 GOSUB 430 :: IF V$="N" THEN 180 ELSE
 IF V$="A" THEN 190
210 DISPLAY AT(2,2)ERASE ALL:WORD$,WORD2
$
220 CALL SPGET(WORD$,W01$):: IF WORD2$<>
"" THEN CALL SPGET(WORD2$,W02$)
230 W=LEN(W01$):: IF WORD2$<>"" THEN W2=
LEN(W02$)
240 DISPLAY AT(4,2):"THE FIRST WORD IS";
W-3:"BYTES LONG":"USE HOW MANY?"
250 ACCEPT AT(7,13)VALIDATE(DIGIT):B ::
IF B>W-3 THEN 250
260 W$=SEG$(W01$,1,2)&CHR$(B)&SEG$(W01$,
4,B)
270 CALL SAY(,W$)
280 GOSUB 430 :: IF V$="N" THEN W$="" ::
 GOTO 250 ELSE IF V$="A" THEN 270
290 IF WORD2$="" THEN 350
300 DISPLAY AT(9,2):"THE SECOND WORD IS"
;W2-3:"BYTES LONG":"USE HOW MANY?"
310 ACCEPT AT(12,13)VALIDATE(DIGIT):B2 :
: IF B2>W2-3 THEN 310
320 W2$=SEG$(W02$,1,2)&CHR$(B2)&SEG$(W02
$,4,B2)
330 CALL SAY(,W2$)
340 GOSUB 430 :: IF V$="N" THEN W2$="" :
: GOTO 300 ELSE IF V$="A" THEN 330
```

```
350 IF WORD2$="" THEN CALL SAY(,W$)ELSE
CALL SAY(,W$,,W2$)
360 GOSUB 430 :: IF V$="N" THEN 210 ELSE
 IF V$="A" THEN 350
370 REM SAVE NAME & SOUND IN ARRAY ELEME
NTS
380 IF WORD2$="" THEN SPEAK$(C,2)=W$ ELS
E SPEAK$(C,2)=W$&W2$
390 DISPLAY AT(14,1):"ENTER PROPER NAME
FOR THIS":"WORD" :: ACCEPT AT(15,6)BEEP:
NAME$ :: IF NAME$="" THEN 390 ELSE SPEAK
$(C,1)=NAME$
400 L=LEN(NAME$&"/"&SPEAK$(C,2)):: IF L>
192 THEN DISPLAY AT(14,1)BEEP:"TOO LONG
TO SAVE !" :: GOSUB 450 :: GOTO 150
410 DISPLAY AT(14,1):"ANY MORE (Y/N) ?"
:: DISPLAY AT(15,1):"" :: ACCEPT AT(14,1
8)BEEP VALIDATE("YN")SIZE(1):V$ :: IF V$
="" THEN 410
420 C=C+1 :: IF V$="N" OR C>40 THEN 470
ELSE 150
430 DISPLAY AT(14,2):"SOUND OK (Y/N/A) ?
" :: ACCEPT AT(14,21)VALIDATE("ANY")SIZE
(1):V$ :: IF V$="" THEN 430
440 DISPLAY AT(14,2):"" :: RETURN
450 FOR DELAY=1 TO 2000 :: NEXT DELAY ::
 RETURN
460 REM SAVE SPEECH WORDS & THEIR SOUNDS
470 OPEN #1:"CS1",FIXED 192,OUTPUT
480 PRINT #1:C-1
490 FOR CT=1 TO C-1
500 PRINT #1:SPEAK$(CT,1)&"/"&SPEAK$(CT,
2)
510 NEXT CT
520 CLOSE #1
530 DISPLAY AT(14,9)ERASE ALL BEEP:"ALL
DONE !"
540 END
```

to the cassette. The cassette file is opened for output at the fixed length of 192.

Line 480 tells the cassette the number of words that will be sent out. One is subtracted from the C variable because when you stop entering the words, C will contain one more than the number of words that you entered.

Line 490 begins the FOR . . . NEXT loop. All the

words and their phoneme sounds will be sent out, one at a time, to the cassette.
Line 500 sends the word and its phoneme string to the cassette. The slash mark separates the phoneme string from the word.
Line 510 continues the loop.
Line 520 closes the file.
Line 530 places the closing message on the screen.

Listing 16-8D should be used if you want to save the words out to disk. The two programs are identical until line 470.

Listing 16-8D

Line 470 erases the screen and asks for a name for the file. If you use a name that is already on the disk, this file will replace it. The computer will loop at this line until a name has been entered. Enter only the name of the file. The computer will add the DSK1. to the name.
Line 480 erases the screen and prints the message that the computer is saving the words to the disk.
Line 490 opens the file to the disk.
Line 500 sends the number of words that will be saved to the disk.
Lines 510-550 send the words in the SPEAK$ array to the disk. The word and its phoneme are not concatenated. The file is closed when all the words have been saved.
Line 560 displays the ending message and the program ends.

Try to make new words with this program. For example, enter the word "message" for the first word and use the first 61 bytes of it. Do not enter a second word. You have just created the word "mess." Or, enter the word "else" for the first word. Use the first 47 bytes. Enter the letter "N" for the second word and use 58 bytes. This combination makes "Allen." Try your own combinations of words to make new words. Sometimes when you shorten the number of bytes that the computer will use, you do not get a word, but a noisy sound.

After you have saved words to the disk or cassette, you may want to use them in another program. The following programs can be used to bring in the information from the cassette or disk. The first program will get the words saved from the cassette, the second from the disk.

Listing 16-9C and 16-9D

Line 130 eliminates the zero element of the string array and sets aside the memory for the array.
Line 140 opens the file to read the cassette.
Line 150 brings in the number that tells the computer how many words were saved in this file.
Line 160 begins the FOR . . . NEXT loop to read in the words.
Line 170 uses the LINPUT command to bring in the word. The word is stored in the temporary string, TEMP$. The POS command is used to locate the slash. The position of the slash is stored in P1.
Line 180 places the contents of the string up to, but not including, the slash in the first element of the string array. This is the word that was created. The character immediately following the slash is the number of characters that make up the phoneme code for the word. The phoneme code is removed from the temporary string and placed in the second part of the array.
Line 190 continues the loop.
Line 200 closes the file.
Lines 210-240 display the words that are in the array and say each word. If you were using these words in your own program, you would not use these lines. Your program would begin here.

The program to bring in the words from the file on disk is similar, except that the string is stored directly into the array. Any program that saves information to the cassette or disk can be brought back into the computer with routines similar to these. In Chapter 14, we saved a character set to the disk or cassette. The character set can be brought in and used in another program. The information must be brought in the same way that it was saved—in a string—then transferred to the string or memory area where it is to be used. In the previous program the information in the string was transferred to a two-dimensional string array. In a character set program, the information would be transferred to the character set.

Listing 16-9C

```
100 REM LISTING 16-9C
110 REM GET SAVED SPEECH - CASSETTE
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 OPTION BASE 1 :: DIM SPEAK$(40,2)
140 OPEN #1:"CS1",FIXED 192,INPUT
150 INPUT #1:CT
160 FOR C=1 TO CT
170 LINPUT #1:TEMP$ :: P1=POS(TEMP$,"/",
1)
180 SPEAK$(C,1)=SEG$(TEMP$,1,P1-1):: L=A
SC(SEG$(TEMP$,P1+1,1)):: SPEAK$(C,2)=SEG
$(TEMP$,P1+2,L)
190 NEXT C
200 CLOSE #1
210 FOR C=1 TO CT
220 DISPLAY AT(12,1)ERASE ALL:"THE WORD
IS ";SPEAK$(C,1):: CALL SAY(,SPEAK$(C,2)
)
230 DISPLAY AT(16,1):"PRESS 'R' TO REPEA
T" :: ACCEPT AT(16,21):V$ :: IF V$="R" T
HEN 220
240 NEXT C
```

Listing 16-9D

```
100 REM LISTING 16-9D
110 REM GET SAVED SPEECH - DISK
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 OPTION BASE 1 :: DIM SPEAK$(40,2)
140 DISPLAY AT(12,1)ERASE ALL:"ENTER NAM
E OF FILE WANTED" :: ACCEPT AT(14,19)BEE
P SIZE(10):FILENAME$
150 OPEN #1:"DSK1."&FILENAME$,SEQUENTIAL
,DISPLAY ,UPDATE,VARIABLE 254
160 INPUT #1:CT
170 FOR C=1 TO CT
180 INPUT #1:SPEAK$(C,1)
190 LINPUT #1:SPEAK$(C,2)
200 NEXT C
210 CLOSE #1
220 FOR C=1 TO CT
230 DISPLAY AT(12,1)ERASE ALL:"THE WORD
```

```
IS ";SPEAK$(C,1):: CALL SAY(,SPEAK$(C,2)
)
240 DISPLAY AT(16,1):"PRESS 'R' TO REPEA
T" :: ACCEPT AT(16,21):V$ :: IF V$="R" T
HEN 230
250 NEXT C
```

Chapter 17
Special Functions

Every memory location in your computer, whether it is RAM, ROM, or GROM, has its own address. These locations contain BASIC, the computer's operating system, and the program that you are using. If you have the Memory Expansion Unit and extra memory, you can use special commands in your Extended BASIC cartridge to look at and change the contents of RAM memory.

HANDLING SPECIFIC MEMORY LOCATIONS

CALL PEEK

To find out what values the computer has stored in a particular location, we need to be able to ask the computer to look for us. We can look by PEEKing at a location. The format for PEEK is:

CALL PEEK(location, variable, variable, etc.)

The location is the memory that we want to look at. The first variable will contain the contents of that memory location. If you want to look at more than one location, each variable following the first, will contain the contents of each subsequent memory location following the one specified. So, if you wanted to see the contents memory locations 500 to 504, you could enter:

CALL PEEK(500,A1,A2,A3,A4,A5)

The variable A1 would contain the contents of memory location 500. A2 would contain 501, A3 502, A4 503, and so on. The contents of any location cannot exceed 255. You can PEEK at any location; however, after location 32767 the computer uses negative memory locations, so memory location 66530 would be accessed by using a negative six as the memory location. Subtract the address that you want to access from 65536 to get the negative address.

CALL INIT

The INITcommand must be used to change the contents of any memory location. This command

tells the computer that you will be accessing memory, loading in a machine language subroutine, or just changing some values in certain memory locations. You need only use this command once. If you are planning to change the contents of memory or load a machine language subroutine, it is good to use this command near the beginning of the program. Once the command is used, it remains active until the Memory Expansion Unit is turned off. The format is CALL INIT.

CALL LOAD

This command will place a new value into a memory location. Its format is:

CALL LOAD(address,byte)

The value of byte will be placed in the memory location specified by address. This command can also be used to load machine language subroutines into memory. In the program in Listing 17-1 (flowcharted in Fig. 17-1) we will use CALL PEEK. After the program runs, we will use CALL INIT and CALL LOAD to show you how the computer's memory can be changed.

Listing 17-1

Line 130 clears the screen.

Line 140 begins the FOR . . . NEXT loop that will display the contents of the memory locations on the screen. The loop begins with a −345. We want to look at memory location 65181. If we subtract 345 from 65536 we arrive at 65181. We do not use a STEP −1 because counting from a negative number whose absolute value is larger to a negative number whose absolute value is less is counting in the positive direction.

Line 150 looks at the location pointed to by the variable ADDRESS. The contents of this location will be placed in the variable BYTE. BYTE$ will contain the character string of BYTE.

Line 160 prints the address that the computer is looking at and the contents of that memory location.

Line 170 looks to see if a key has been pressed. If a key has not been pressed the S variable will be zero. We are not interested in what key was pressed, just if a key was pressed. Pressing a key will pause the program and hold the information on the screen.

Line 180 is a delay loop. This temporarily stops the program from printing the contents of the memory locations on the screen. After the loop, the computer will be directed back to line 170. As long as a key is pressed, the computer will loop between these two lines. This gives you a chance to study or copy the information on the screen.

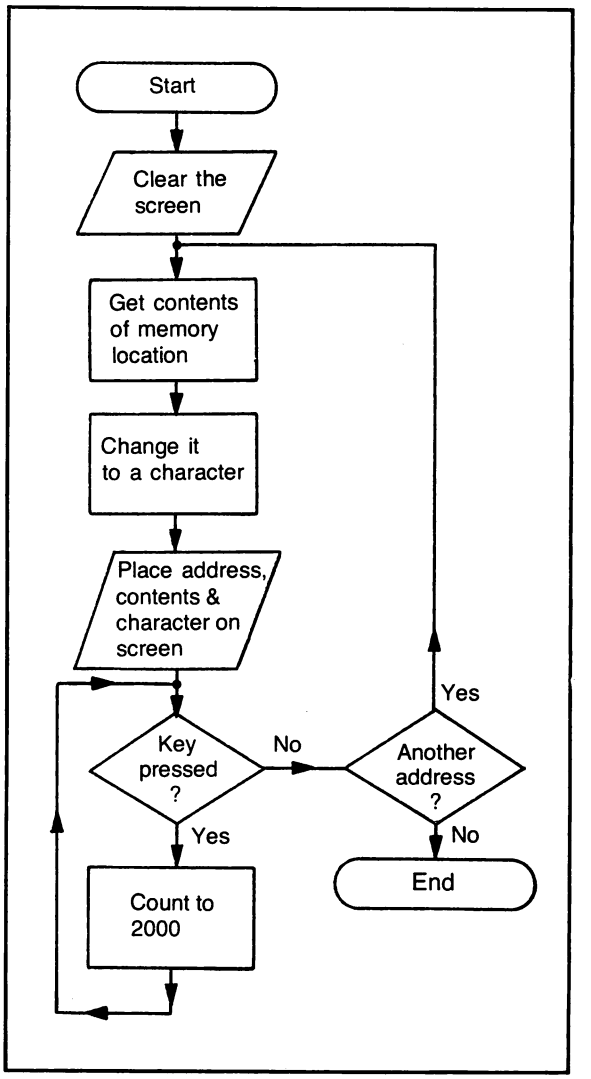


Fig. 17-1. Flowchart for Listing 17-1 Peeking at a Program.

Listing 17-1

```
100 REM LST-17-1
110 REM PEEKING AT PROGRAM
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR
140 FOR ADDRESS=-345 TO -24
150 CALL PEEK(ADDRESS,BYTE):: BYTE$=CHR$
(BYTE)
160 PRINT ADDRESS+65536;TAB(10);BYTE;TAB
(18);BYTE$
170 CALL KEY(0,K,S):: IF S=0 THEN 190
180 FOR DELAY=1 TO 2000 :: NEXT DELAY ::
 GOTO 170
190 NEXT ADDRESS
200 END
210 REM END OF LISTING
```

Line 190 continues the loop until all the memory addresses have been looked at.

Now, in the direct mode, enter

CALL INIT

so that we can use the CALL LOAD command. Again, from the direct mode, enter:

MESSAGE$=" **PRESTO CHANGO**"

Be sure that there is a space before the first asterisk. Press the ENTER key. Now in the direct mode enter:

FOR Z=1 TO LEN(MESSAGE$)::
Y=ASC(SEG$(MESSAGE$,Z,1))::CALL
LOAD(-56+Z,Y)::NEXT Z

Press the ENTER key, then, LIST the program. Line 110 should now read:

110 REM **PRESTO CHANGO**

You have just changed the program by loading new values into the memory locations that held line 110. By knowing where the computer stores the program and other information, you can change it whenever you need to. If you look closely at the codes that this program printed on the screen, you will notice that the program is stored in the highest memory locations available. The highest 25 bytes of memory are used by the computer. Then the program is stored. Just before the program are the program line numbers with two numbers separating the line numbers. These two numbers are the address of where that line is stored in memory.

DEF

This command allows you to create your own functions. You can use it in a program where the computer will be using a certain formula several times. You could make this formula a subroutine, have the computer go to it when it needs to use that formula and return, or you can make the formula your own function with its own name. Every time the computer sees that name, it will use the formula. The program in Listing 17-2 (flowcharted in Fig. 17-2) does just that. The T variable becomes a formula to come up with a value for a pitch or tone. When the computer uses T in the SOUND command, it computes its value automatically.

Listing 17-2

Line 130 erases the screen and places the direc-

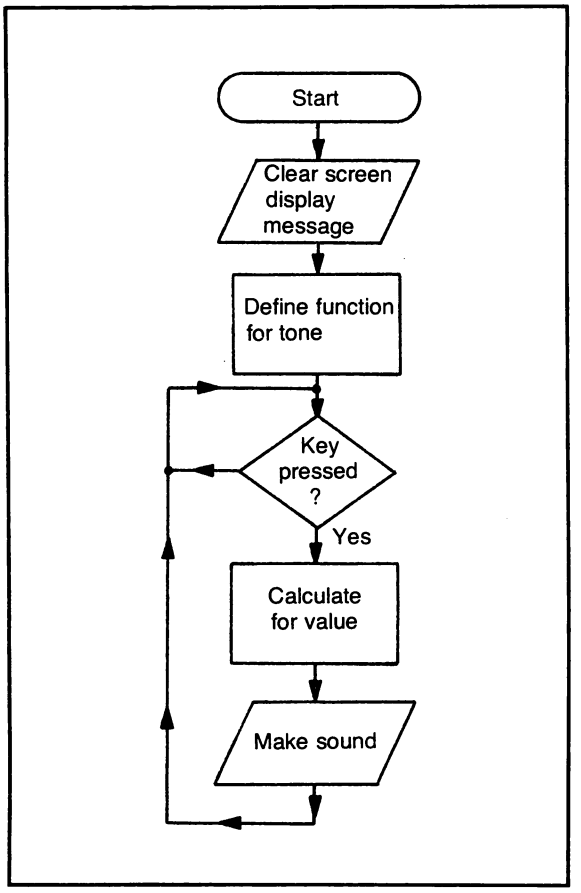


Fig. 17-2. Flowchart for Listing 17-2 Keyboard Tones.

tions on the screen. Adjust the volume on your television or monitor to a comfortable level, then press any key.

Line 140 defines a function. In this program we use the standard formula for finding the frequency of a note. Every time the computer finds T in the program it will use this formula to find out what the value of T is. It will use this value in that program line.

Line 150 uses the CALL KEY command to see if a key has been pressed. If the S variable is zero, a key has not been pressed and the computer will loop at this line until a key has been pressed.

Line 160 subtracts 31 from the value of K. By subtracting 31, we can make a tone from any key whose value is between 32 and 96. Any value less than 32 would become less than one. You cannot raise the value to a negative number and get a value that the computer can play. Any value higher than 96 will produce a tone that is too high for most people to hear.

Line 170 makes the tone based on the value of T. The computer will use the formula in line 140 to arrive at the value of T. We use a negative number for the duration so that the computer will not pulse if the same key is pressed for an extended period of time. If a new key is pressed before the computer has finished playing this tone, it will make a smooth change to the new tone.

Listing 17-2

```
100 REM LISTING 17-2
110 REM KEYBOARD TONES
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 DISPLAY AT(11,8)ERASE ALL:"ADJUST VO
LUME" :: DISPLAY AT(13,8):"PRESS ANY KEY
"
140 DEF T=110*(2^(1/12))^K
150 CALL KEY(0,K,S):: IF S=0 THEN 150
160 K=K-31 :: IF K<1 OR K>64 THEN 150 !
TONES MUST BE HEARABLE
170 CALL SOUND(-1000,T,0)! PLAY IT
180 GOTO 150
```

Line 180 sends the computer back to line 150 for a new key value.

ELIMINATING THE ENTER KEY

CALL KEY

We have been using the CALL KEY command throughout this book. Whenever we've wanted to be able to check the key that has been pressed without waiting for the ENTER key, we used the CALL KEY. We also used the CALL KEY command to see if the fire button on the joystick had been pressed.

CALL KEY can not only check to see whether or not any key has been pressed, but can check for specific keys as well. The various format to check keys are listed below:

CALL KEY(0,K,S)	any key
CALL KEY(1,K,S)	left side of keyboard & joystick 1
CALL KEY(2,K,S)	right side of keyboard & joystick 2
CALL KEY(3,K,S)	lowercase value regardless of whether key is upper or lowercase.
CALL KEY(4,K,S)	Pascal Codes
CALL KEY(5,K,S)	BASIC mode

The K variable will hold the ASCII value of the key that has been pressed. S is the status of the command. The value of S will be zero if no key has been pressed. The value will be one if a new key has been pressed and negative one if the same key has been pressed as the last time that the computer used this command. In the program in Listing 17-3 (flowcharted in Fig. 17-3) the computer will use CALL KEY to accept entries from the keyboard. Keep your ALPHA LOCK key up so that you can enter upper or lowercase keys. You can also press any two-key combination (CTRL and W, FCTN and 1) and see what codes will be returned. Do not press FCTN and 4 (clear) or FCTN and 8 (quit). They will leave the program.

Listing 17-3

Lines 130-140 clear the screen and display the message and the format for the CALL KEY command.

Line 150 begins the FOR . . . NEXT loop that displays the six different ways that the CALL KEY command can be used. The CALL KEY command is used with the UNIT variable set by the FOR. . . NEXT loop. The value of the key that has been pressed is placed in the variable RTURN. When a key has been pressed, the value of STATUS will be a one. STATUS will be a zero when no key is being pressed.

Line 160 checks the value of STATUS when the UNIT value is zero. If no key is being pressed, the computer goes back to line 150 and waits until a key has been pressed.

Lines 170-180 print the UNIT value, the value of the key, and the STATUS on the screen. If the key

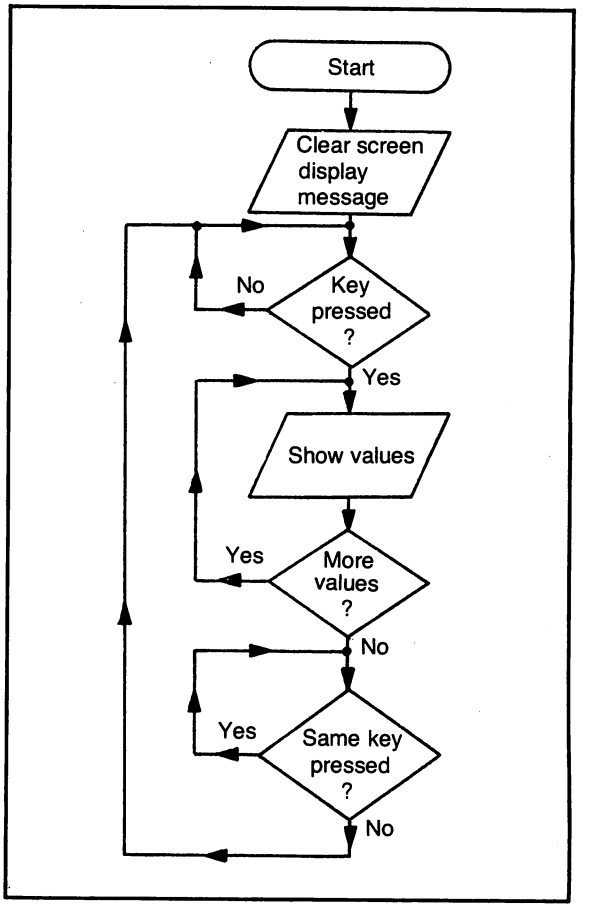


Fig. 17-3. Flowchart for Listing 17-3 Keyboard-Kapers.

Listing 17-3

```
100 REM LISTING 17-3
110 REM KEYBOARD-KAPERS
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 DISPLAY AT(2,6)ERASE ALL:"HOLD DOWN
A KEY" :: DISPLAY AT(3,2):"UNTIL CODES S
TOP PRINTING" :: DISPLAY AT(6,1):"FORMAT
:"
140 DISPLAY AT(8,1):"CALL KEY(UNIT,RETUR
N,STATUS)"
150 FOR UNIT=0 TO 5 :: CALL KEY(UNIT,RTU
RN,STATUS)
160 IF UNIT=0 AND STATUS=0 THEN 150
170 DISPLAY AT(2*UNIT+11,1):"CALL KEY(";
UNIT;TAB(14);",";
180 DISPLAY AT(2*UNIT+11,16):RTURN;TAB(2
1);",";TAB(25);STATUS;TAB(28);")"
190 NEXT UNIT
200 CALL KEY(0,K,S):: IF S=-1 THEN 200
210 GOTO 150
```

cannot be read for a particular unit value, for example, the computer is reading the right side of the keyboard and the key that you are pressing is on the left side, the value of STATUS will be zero. In the next line, the STATUS value will be a one—new key pressed—when, in fact, it is the same key that you have been pressing. If the computer uses the value of the key in two consecutive lines, the STATUS value in the second line will be negative one.

Line 190 continues the routine until all six variations of this command are displayed.

Line 200 uses the CALL KEY command again. This time it checks to see if the key is still being pressed. If it is, the STATUS will be negative one. The computer will loop at this line until the key is no longer being pressed.

Line 210 sends the computer back to line 150 to set another key.

PRINTING IN COLUMNS

We have been using commas and semicolons with spaces to print information in certain patterns on the screen. The computer can have a pattern stored in its memory and then use that pattern within a program whenever it needs to by using the IMAGE with the PRINT USING or DISPLAY USING commands.

The IMAGE command tells the computer how you would like the information printed. It can be used for numbers, numeric variables, or string variables. The IMAGE that you want to use must be in a program line before the line that it will be used in. Words can be used in the IMAGE statement. Here are some samples of how to use IMAGE:

100 IMAGE $###.##—prints the dollar sign and the amount up to three digits preceding the decimal.

100 IMAGE YOU'RE BALANCE IS $####.##—prints the words "you're balance is" and the amount up to four digits preceding the decimal.

100 IMAGE MEMO TO #####—prints a word

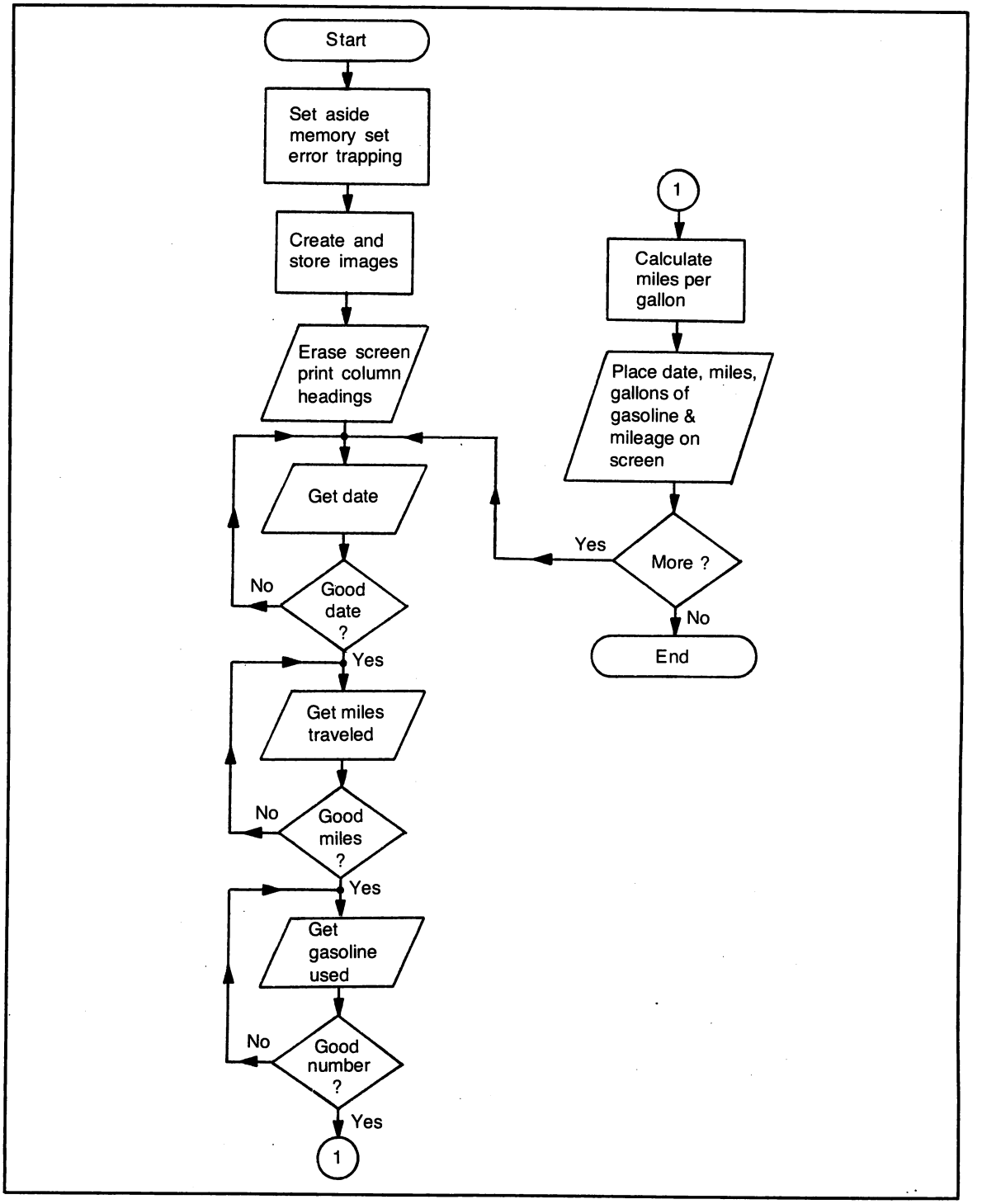


Fig. 17-4. Flowchart for Listing 17-4 Using.

Listing 17-4

```
100 REM LISTING 17-4
110 REM USING
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 DIM DATE$(52),MILES(52),GASOLINE(52)
,MILEAGE(52)
140 ON WARNING NEXT
150 IMAGE ########
160 IMAGE ####.#
170 IMAGE ##.#
180 IMAGE ##.#
190 DISPLAY AT(1,8)ERASE ALL:"MILEAGE RE
PORT" :: DISPLAY AT(3,3):"DATE";TAB(12);
"MILES";TAB(20);"GAS";TAB(26);"MPG"
200 DISPLAY AT(4,1):RPT$("-",28)
210 ROW=5 :: FOR C=1 TO 52
220 DISPLAY AT(23,1):"ENTER DATE (03/04/
83) :" :: ACCEPT AT(24,5)BEEP SIZE(8):TE
MP$ :: IF TEMP$="" THEN 220
230 IF LEN(TEMP$)<>8 THEN 220 ELSE DATE$
(C)=TEMP$
240 DISPLAY AT(23,1):"ENTER MILES TRAVEL
ED :" :: ACCEPT AT(24,5)BEEP VALIDATE(NU
MERIC):TEMP :: IF TEMP<1 OR TEMP>9999.9
THEN 240
250 MILES(C)=TEMP
260 DISPLAY AT(23,1):"ENTER GASOLINE USE
D :" :: ACCEPT AT(24,5)BEEP VALIDATE(NUM
ERIC):TEMP :: IF TEMP<.1 OR TEMP>99.9 TH
EN 260
270 GASOLINE(C)=TEMP :: MILEAGE(C)=MILES
(C)/GASOLINE(C)
280 DISPLAY AT(ROW,1):USING 150:DATE$(C)
290 DISPLAY AT(ROW,11):USING 160:MILES(C
)
300 DISPLAY AT(ROW,19):USING 170:GASOLIN
E(C)
310 DISPLAY AT(ROW,25):USING 180:MILEAGE
(C)
320 ROW=ROW+1 :: IF ROW=22 THEN ROW=5
330 NEXT C
```

or number up to five characters long.

100 IMAGE INVOICE ####—prints a word or number up to four characters long.

The computer can refer to the patterns in the above examples by line number. The program in Listing 17-4 (flowcharted in Fig. 17-4) contains examples of these commands.

Listing 17-4

Line 130 sets aside memory for the strings. The computer will allow up to 52 entries.

Line 140 traps errors.

Lines 150-180 set the IMAGEs for the numbers that will be used in this program.

Line 190 prints the top heading on the screen. These set the column for the entries.

Line 200 uses the RPT$ to print the line across the screen.

Line 210 sets the ROW variable for five. This is the row that the first entry will be printed on. The FOR . . . NEXT loop begins on this line.

Line 220 places an example for the date on the lower line and waits for the date to be entered. The date is placed in TEMP$.

Line 230 checks the length of the string. If it is not 8 characters long, the computer will go back to line 220 for another entry; otherwise it will be placed in DATE$, where the C variable is pointing.

Line 240 asks for the miles traveled. If the miles traveled is less than one or greater that 9999.9 the computer will loop at this line.

Line 250 places the miles traveled in the MILES array.

Line 260 asks for the number of gallons of gasoline used. If less than one tenth of a gallon or more than 99.9 gallons were used, the computer loops at this line.

Line 270 places the gallons used in the GASOLINE array. The mileage is determined by dividing the miles driven by the gallons of gasoline used.

Lines 280-310 places the information entered and the mileage on the screen. Each line uses a different IMAGE for the information that it is placing on the screen. The line number after USING refers to the IMAGE that the computer will use. The information will be printed in straight columns whether each entry in that column is the same length or not.

Line 320 adds one to the value of ROW so that the next entry can be printed on the next line. If the value of ROW is 22, then the computer will begin printing at row five again.

Line 330 continues the loop until 52 entries have been made.

Each IMAGE statement must be in a different line. There cannot be any other command or statement on that line. If you enter another command on the line before the IMAGE statement, the computer will not accept that line and print a syntax error on the screen. Putting a statement or command on the same line after the IMAGE statement will cause the computer to print that statement or command as part of the IMAGE message when the IMAGE statement is used in the program. The RESEQUENCE command will renumber the references to the IMAGE lines in the program so your program will execute properly.

The computer will use the DISPLAY USING or PRINT USING without the IMAGE statement if the image is contained in the program line.

```
100 DISPLAY AT(2,3):USING ###.#:A
```

The contents of A will be printed using the image of ###.#. A line number was not referenced and an IMAGE statement was not used. The image can also be placed in a string, as shown below:

```
100 A$="####.##"
110 PRINT USING A$:B
```

Chapter 18
Advanced Programming Skills

Now that you are confidently programming on your TI-99/4A computer, you may want to give your programs a more professional look. Routines may be taking too long; other programs could use more color or better graphics. Some of these problems can be solved by using machine language subroutines within your BASIC programs, others by using sprites.

MACHINE/ASSEMBLY LANGUAGE

Before writing a machine language subroutine, you must have some knowledge of the instructions the microprocessor follows. This chapter is not designed to teach you about machine language. It will only give you a general explanation of how your TI-99/4A can use a machine language subroutine within a BASIC program.

In machine code, each instruction is a number. This number tells the microprocessor within the computer what it should do. It would be tedious to write machine language programs using BASIC. There is a special editor/assembler available for your TI-99/4A designed for writing machine language programs. You use the editor/assembler to write the program in assembly language. Once you have written the program in assembly language, it will be converted to machine code by the editor/assembler.

You can save your machine language this program to disk or cassette. The name by which you want to access your machine language program must be specified with a DEF statement in the assembly language program. From BASIC, you can load this machine language program with the CALL LOAD command as follows:

CALL LOAD(DSK1.name")

Once you have loaded the machine language program into the computer, you can access it from BASIC with the CALL LINK command as follows:

CALL LINK("subprogram name")

The name can be followed by variable values that

are to be passed to the machine language program similar to the CALL SUB command. The computer would search its table to find out where this machine language program is stored and execute it. After it is executed, it will return to BASIC. If the machine language program has not been loaded into memory, an error message will appear and the program will crash.

Since the editor/assembler, Memory Expansion Unit, and memory is needed to utilize these commands, it is beyond the scope of this book to present viable examples of these commands.

USING SPRITES

Sprites add interesting and creative effects to programs. Sprites are independent of the character set in as much as they can be placed anywhere on the screen, can move independently of each other and characters, and can be one of two sizes. With your TI-99/4A, you can place up to 28 sprites on your screen.

A sprite is created with the same command that a character is created—CALL CHAR. A character that will be a sprite is created and placed in the normal character set. This character is then used as a sprite with the CALL SPRITE command. The format is as follows:

CALL SPRITE(number,character number,color,row,column)

After the row and column, you can also add the speed at which the sprite will move and the direction—horizontally, vertically, or a combination—to move on the diagonal.

The row can be any number from 1 to 192 and the column can be any number from 1 to 256. This is a dot position rather than an actual row and column number, giving you greater flexibility as to where you want to place the sprites.

A sprite can be placed over a character that is on the screen. The ability to place one sprite over another sprite or character is called a *priority*. Characters have a lower priority than sprites, so a sprite can cover a character. The sprite with the lowest number will cover the other sprite or portion of the sprite if it tries to occupy the same area of the screen.

The program in Listing 18-1 creates a space ship (Fig. 18-1) and uses it as a sprite. The sprite is placed on the screen as a character. You can enter different colors and positions to place the sprite on the screen. The upper left corner of the sprite will be placed in the row and column number that you enter. The program will allow you to place all 28 sprites.

Listing 18-1

Line 130 protects the program against null entries for the variables. It also creates a sprite (Fig. 18-2) and places it in an undefined character location. You can use any defined character as a sprite. You can place your own characters into the undefined character space and use them as sprites.

Line 140 erases the entire screen. The ERASE ALL option and the CALL CLEAR command does not affect the sprites. Only the characters will be erased. The format for the CALL SPRITE command is printed on the screen.

Lines 150-160 print the remaining portion of the CALL SPRITE command on the screen.

Line 170 places the empty command and the message on the screen. You can enter any values that you like to create your own sprites.

Line 180 beeps and waits for a sprite number. This number can be between 1 and 28. The number sign (#) is already on the screen. You need only

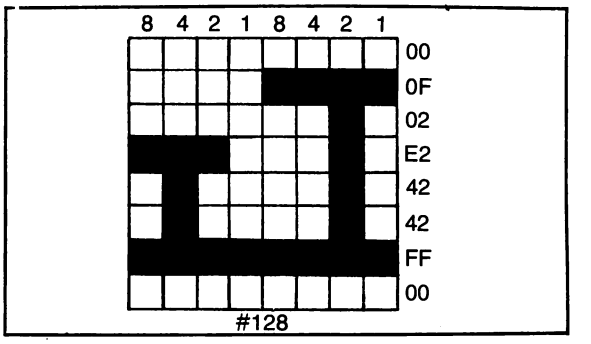


Fig. 18-1. Character for Listing 18-1 Creating Sprites.

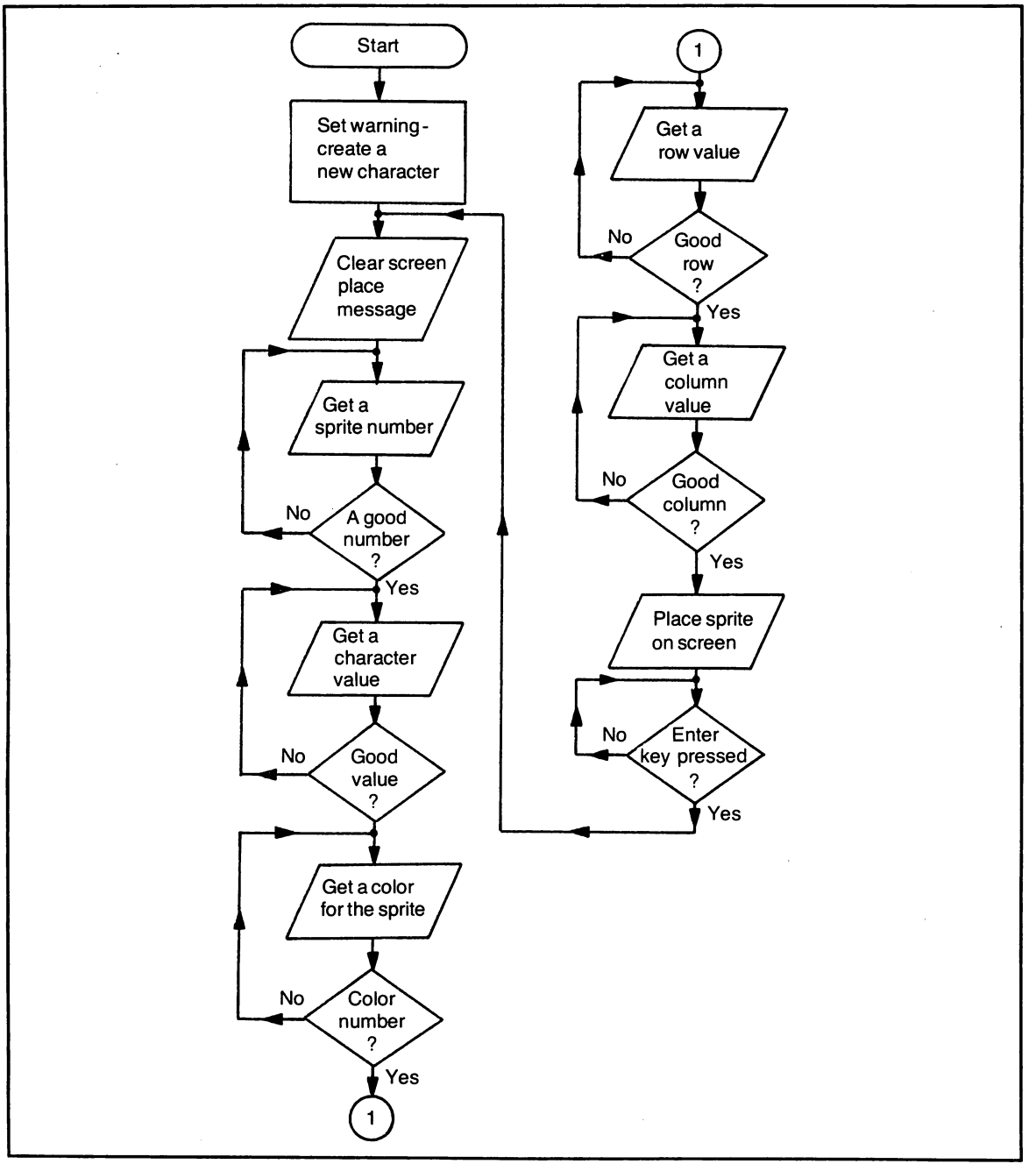


Fig. 18-2. Flowchart for Listing 18-1 Creating Sprites.

enter the sprite number.

Line 190 waits for the ASCII value of the character that you want to use as a sprite. If you would like the ship to appear on the screen, enter 128. Otherwise, enter any value that you would like between 32 and 143.

Listing 18-1

```
100 REM LISTING 18-1
110 REM CREATING SPRITES
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 ON WARNING NEXT :: CALL CHAR(128,"00
0F02E24242FF")
140 DISPLAY AT(2,1)ERASE ALL:"FORMAT :"
:: DISPLAY AT(4,1):"CALL SPRITE(#SPRITE
NUMBER,"
150 DISPLAY AT(5,3):"CHAR, VALUE,SPRITE-
COLOR,"
160 DISPLAY AT(6,6):"DOT-ROW,DOT-COLUMN)
"
170 DISPLAY AT(23,1):"CALL SPRITE(#   ,
 ,   ,    ," :: DISPLAY AT(24,1):"    )";TA
B(17);"ENTER VALUES"
180 ACCEPT AT(23,14)BEEP VALIDATE(DIGIT)
SIZE(2):SN :: IF SN<1 OR SN>28 THEN 180
190 ACCEPT AT(23,17)BEEP VALIDATE(DIGIT)
SIZE(3):CV :: IF CV<32 OR CV>143 THEN 19
0
200 ACCEPT AT(23,21)BEEP VALIDATE(DIGIT)
SIZE(2):SC :: IF SC<1 OR SC>16 THEN 200
210 ACCEPT AT(23,24)BEEP VALIDATE(DIGIT)
SIZE(3):DR :: IF DR<1 OR DR>192 THEN 210
220 ACCEPT AT(24,1)BEEP VALIDATE(DIGIT)S
IZE(3):DC :: IF DC<1 OR DC>256 THEN 220
230 CALL SPRITE(#SN,CV,SC,DR,DC):: DISPL
AY AT(24,17):"PRESS ENTER"
240 CALL KEY(0,K,S):: IF S=0 THEN 240 EL
SE IF K<>13 THEN 240 ELSE 140
```

Line 200 accepts the color number. This will be the color of your sprite. Unlike characters, you can only set the foreground color and the sprite. Enter a number between 1 and 16.

Line 210 waits for the row number. There are 192 rows on the screen for your sprite. Enter a number between one and 192.

Line 220 places the number you enter in DC. This variable will set the column for the sprite. The column can be between one and 256.

Line 230 uses the CALL SPRITE command. The sprite will be set per your entries. The sprite will be displayed on the screen at the row and column that you specified. If you do not see a sprite on the screen, the color of the sprite may be blending into it, or the row and/or column number is so low that the sprite is off the screen. Some televisions or monitors will display the entire screen, others will not show the right and left edges or the top and bottom rows.

Line 240 uses the CALL KEY command. The computer will loop at this line until the enter key is pressed. It will then go back to line 140 for another entry.

The program will continue until you press the clear key. You can move any sprite that you have

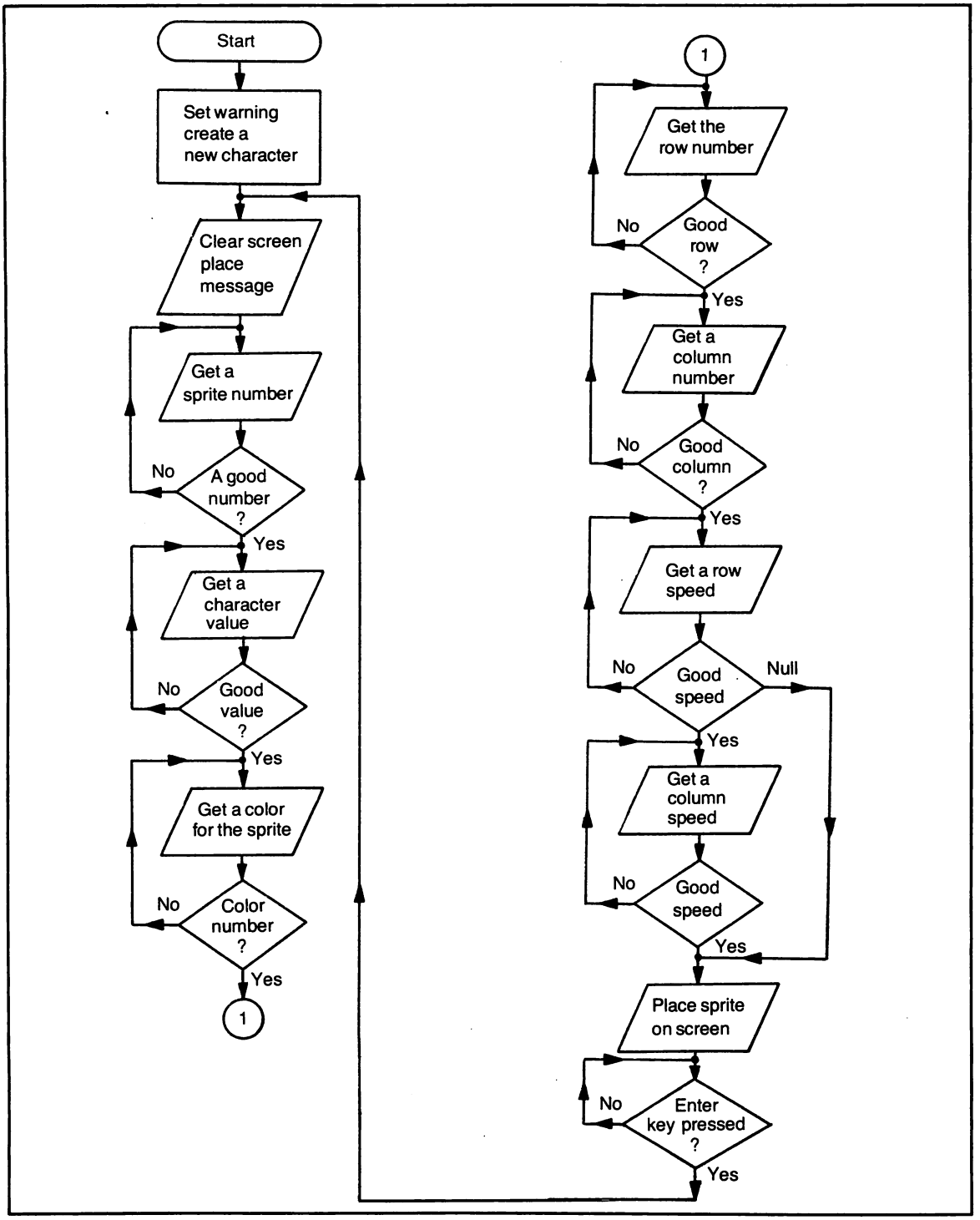


Fig. 18-3. Flowchart for Listing 18-2 Moving Sprites.

Listing 18-2

```
100 REM LISTING 18-2
110 REM MOVING SPRITES
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 ON WARNING NEXT :: CALL CHAR(128,"00
0F02E24242FF")
140 DISPLAY AT(2,1)ERASE ALL:"FORMAT :"
:: DISPLAY AT(4,1):"CALL SPRITE(#SPRITE
NUMBER,"
150 DISPLAY AT(5,3):"CHAR, VALUE,SPRITE-
COLOR,"
160 DISPLAY AT(6,5):"DOT-ROW,DOT-COLUMNI
,ROW-" :: DISPLAY AT(7,7):"SPEED,COLUMN-
SPEED])"
170 DISPLAY AT(23,1):"CALL SPRITE(#   ,
 ,   ,    ," :: DISPLAY AT(24,1):"    [,
 ,     ])ENTER VALUES"
180 ACCEPT AT(23,14)BEEP VALIDATE(DIGIT)
SIZE(2):SN :: IF SN<1 OR SN>28 THEN 180
190 ACCEPT AT(23,17)BEEP VALIDATE(DIGIT)
SIZE(3):CC :: IF CC<32 OR CC>143 THEN 19
0
200 ACCEPT AT(23,21)BEEP VALIDATE(DIGIT)
SIZE(2):SC :: IF SC<1 OR SC>16 THEN 200
210 ACCEPT AT(23,24)BEEP VALIDATE(DIGIT)
SIZE(3):DR :: IF DR<1 OR DR>192 THEN 210
220 ACCEPT AT(24,1)BEEP VALIDATE(DIGIT)S
IZE(3):DC :: IF DC<1 OR DC>256 THEN 220
230 ACCEPT AT(24,6)BEEP VALIDATE(NUMERIC
)SIZE(4):TEMP$ :: IF TEMP$="" THEN RV,CV
=0 :: GOTO 270
240 RV=VAL(TEMP$):: IF RV<-128 OR RV>127
 THEN 230
250 ACCEPT AT(24,11)BEEP VALIDATE(NUMERI
C)SIZE(4):TEMP$ :: IF TEMP$="" THEN 250
260 CV=VAL(TEMP$):: IF CV<-128 OR CV>127
 THEN 250
270 CALL SPRITE(#SN,CC,SC,DR,DC,RV,CV)::
 DISPLAY AT(24,17):"PRESS ENTER"
280 CALL KEY(0,K,S):: IF S=0 THEN 280 EL
SE IF K<>13 THEN 280 ELSE 140
```

placed on the screen by setting the row and column numbers to a new position. You can change the sprite by using another character number.

The sprites that you entered, did not move on the screen. In order to get movement, you need to enter the velocity or speed of the sprite. The number can be negative or positive depending on which direction you want the sprite to move in.

LEFT	-128 to -1
RIGHT	1 to 127
UP	-128 to -1
DOWN	1 to 127

The closer the number is to zero, the slower the sprite movement. A zero for both directions keeps the sprite in one position on the screen. A zero for one direction moves the sprite in a straight line in the other direction; for instance, a zero for horizontal moves the sprite vertically. A number for both directions moves the sprite diagonally. The program in Listing 18-2 (flowcharted in Fig. 18-3) allows you to add the horizontal and vertical speed to the sprites.

Listing 18-2

Lines 130-220 are the same as in the previous programs.

Line 230 validates the speed for negative or positive numbers. The speed of the sprite can be negative or positive. It is stored in a string so that the string can be tested for a value or a null string. The speed is optional in the CALL SPRITE command. You do not have to enter a value for it. If you just press the ENTER key, the computer will place the sprite on the screen and it will not move.

The RV and CV variables will be set to zero.

Line 240 places the value of the string in the RV variable. If the value entered is not between -128 and 127, the computer will go back to line 230 for another value.

Line 250 accepts a value for the column speed. This time if no value is entered, the computer will remain at this line until a value is entered.

Line 260 places the value entered into the CV variable. The computer checks to see if the number entered is valid. If it isn't the computer will go back to line 250 for another number.

Line 270 places the sprite on the screen. If the RV and CV variables are set, the sprite will move on the screen. If they have not been set, the sprite will remain in one place on the screen.

Line 280 waits for ENTER to be pressed. When it is, the computer goes back to line 140 for another sprite.

CALL MOTION

There may be times where you do not want to set a sprite into motion when you place it on the screen. You want the player or program to control when the sprite should be moved. This is when the call motion command can be used. Load in the cake program from Chapter 16. We will add sprites to this program. (See Listing 18-3.) While the candles are lit, the balloons remain still. After the candles go out, the balloons start to move on the screen.

Listing 18-3 (flowcharted in Fig. 18-4)

Line 212 places three new characters in the character set beginning with character 128. These three characters are the balloons. (See Fig. 18-5)

Line 214 sets the variable SP to the first balloon character number. The FOR. . .NEXT loop places a balloon in a sprite. The color along with the row and column that the sprite will be displayed at are chosen randomly. When SP reaches 131, it is reset to 128.

Line 216 continues the loop.

Line 240 blows the balloons up on the screen. When the music is almost done, the balloons change size with the MAGNIFY command. The two in parentheses makes the balloons twice the size they would have been in the normal mode. The FOR . . . NEXT loop uses the CALL MOTION command to get the sprites moving on the screen. The computer chooses a random number for the direction and speed of each sprite. By multiplying the number chosen by a -1, the computer can only make the balloons go up on the screen. Ten is

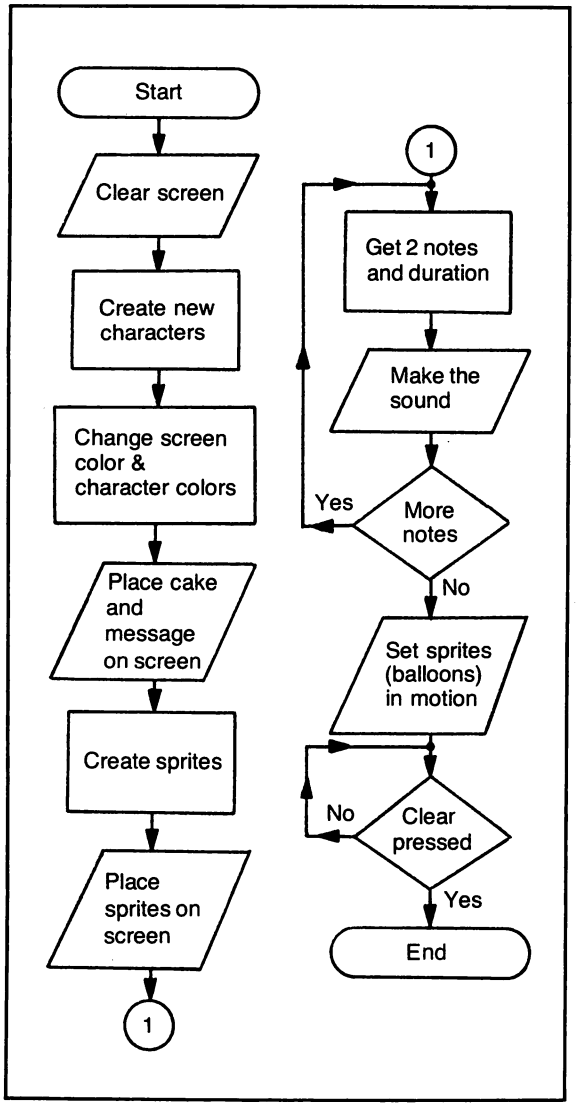


Fig. 18-4. Flowchart for Listing 18-3 Cake & Balloons.

subtracted from the column value. If the new value is negative, the balloon will float to the left. If the value is positive, the balloon will float right.

Line 245 keeps the computer in a loop until the clear key is pressed.

You may notice that sometimes part of a balloon disappears from the screen. The TI-99/4A can only have four sprites in one row. When more than four sprites are in the same row, only the first four will be visible on the screen.

CALL LOCATE

This command will move the sprite on the screen. It can move any sprite to any location. In the next program we will move an arrow across the bottom of the screen. By pressing the D or S key the arrow moves one column at a time to the right or left. But, if you press the F or A key, the arrow will move much faster because it is actually moving five columns to the right or left.

Listing 18-4

Line 130 clears the screen, creates the new characters and sets the magnification for the sprite at two.

Line 140 gets COL to 125. This is the column that the sprite will be printed at. The CALL SPRITE command is used to place the sprite on the screen.

Line 150 checks to see if a key has been pressed. The computer will loop at this line until a key is pressed. The first value of the key command is one. Only the keys on the left side of the keyboard will change the status value.

Line 160 checks the value of K for a two. If the S has

Listing 18-3

```
100 REM LISTING 18-3
110 REM CAKE & BALLOONS
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 RANDOMIZE :: CALL CLEAR :: FOR I=99
TO 109 :: READ C$ :: CALL CHAR(I,C$):: N
EXT I
140 FOR I=112 TO 116 :: READ C$ :: CALL
```

```
CHAR(I,C$):: NEXT I
150 FOR I=120 TO 121 :: READ C$ :: CALL
CHAR(I,C$):: NEXT I
160 FOR I=1 TO 9 :: READ C,C$ :: CALL CH
AR(C,C$):: NEXT I
170 CALL COLOR(9,12,10,10,9,10,11,3,1):: 
 CALL SCREEN(12)
180 DISPLAY AT(10,12):"deccfs" :: DISPLA
Y AT(11,12):"hiiiiJ" :: DISPLAY AT(12,12
):"111111"
190 DISPLAY AT(13,11):"pk1111mt" :: DISP
LAY AT(14,11):"qrrrrrrs"
200 DISPLAY AT(9,14):"xx"
210 DISPLAY AT(6,11):"H A P P Y" :: DISP
LAY AT(17,8):"B I R T H D A Y"
212 CALL CHAR(128,"183C7E7E3C18000183C3
C7E7E3C3C18187EFFFF7E3C3C18")
214 SP=128 :: FOR X=1 TO 28 :: CALL SPRI
TE(#X,SP,RND*6+2,RND*30+150,RND*200+10):
: SP=SP+1 :: IF SP=131 THEN SP=128
216 NEXT X
220 FOR I=1 TO 26 :: READ D,N,N1 :: CALL
 SOUND(D,N,0,N1,0):: NEXT I
230 DISPLAY AT(9,14):"yy"
240 CALL MAGNIFY(2):: FOR X=1 TO 28 :: C
ALL MOTION(#X,RND*10*-1-1,RND*20-10):: N
EXT X
245 GOTO 245
250 DATA FF,FFFFFCF0C08,FF8,FF01,FFFF3F0
F0301
260 DATA 40201F,0000FF,0204FC,0000000000
80C0E,00,0000000000010307
270 DATA 00000003070F0F07,0301,FFFFFF,C0
8,000000C0E0F0F0E,0010029012929292,00000
01012929292
280 DATA 65,7E66667E6666F7,66,FC66667C66
66FC,68,FC6666666666FC,72,F766667E6666F7
290 DATA 73,3C18181818183C,80,FC66667C60
60F,82,FC66667C666677,84,7E55551818183C
300 DATA 89,EF6666667E067E
310 DATA 250,262,220,250,262,220,500,294
,220,500,262,220,500,349,262,750,330,262
320 DATA 250,262,196,250,262,196,500,294
```

```
,196,500,262,196,500,392,262,750,349,262
330 DATA 250,262,220,250,262,220,500,523
,349,500,440,349,250,349,262,250,349,262
,500,330,262,750,294,294
340 DATA 250,466,349,250,466,349,500,440
,349,500,349,262,500,392,262,750,349,262
```

been pressed, one will be subtracted from the value of COL.

Line 170 checks for the D key. If it has been pressed, one will be added to COL so that the arrow can be moved to the right.

Line 180 checks for the A key. This key will make the arrow move to the left faster by subtracting five from the variable COL.

Line 190 will add five to the value of COL if the F key is pressed.

Lines 200-210 check to see if the arrow will be off the screen. If it will, the variable will be reset for the edge of the screen.

Line 220 uses the CALL LOCATE command to place the sprite on the screen. Each sprite can only occupy one position on the screen. When we place the arrow in a new position, the computer erases the arrow before it places it in the new position. The movement is much smoother than when we moved characters by erasing one, then reprinting it.

Line 230 sends the computer back to line 150 for another key.

CALL MAGNIFY

In the Cake and Balloons program, we used the CALL MAGNIFY command to make the balloons larger on the screen. This command makes the sprite four times as large as the normal character. The four different CALL MAGNIFY options listed below:

CALL MAGNIFY(1)	normal size
CALL MAGNIFY(2)	four times normal size
CALL MAGNIFY(3)	normal size 4×4 sprite
CALL MAGNIFY(4)	four times 4×4 sprite

Since we already used the CALL MAGNIFY(2), the program in Listing 18-5A will demonstrate CALL MAGNIFY(4). With the CALL MAGNIFY(3) or CALL MAGNIFY(4), the computer takes four sequential characters and places them on the screen in a square. If the MAGNIFY(3) is used, the characters will be the normal size, but because the square is 2×2, the character could be redefined and four times as large as a normal character. If the MAGNIFY(4) is used, the charac-

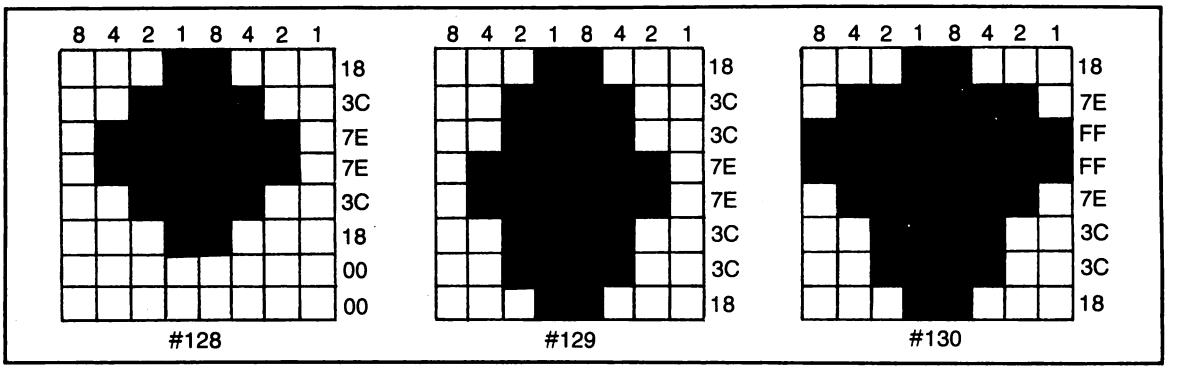


Fig. 18-5. Characters for Listing 18-3 Cake & Balloons.

Listing 18-4

```
100 REM LISTING 18-4
110 REM MOVING THE ARROW
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: CALL CHAR(128,"1028545
410101010"):: CALL MAGNIFY(2)
140 COL=125 :: CALL SPRITE(#1,128,5,150,
COL)
150 CALL KEY(1,K,S):: IF S=0 THEN 150
160 IF K=2 THEN COL=COL-1
170 IF K=3 THEN COL=COL+1
180 IF K=1 THEN COL=COL-5
190 IF K=12 THEN COL=COL+5
200 IF COL<8 THEN COL=8
210 IF COL>244 THEN COL=244
220 CALL LOCATE(#1,150,COL)
230 GOTO 150
```

LISTING 18-5A

```
100 REM LISTING 18-5A
110 REM THE DUCKS
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: M=3
140 CALL CHAR(128,"060F1F37FF3F4F06060F1
F3F3F1F0F0700008C0C08F1C3870E0FCFFFEFCF
8F0")
150 CALL SPRITE(#1,128,5,100,120)
160 DISPLAY AT(10,8):"MAGNIFY AT";M :: C
ALL MAGNIFY(M)
170 CALL KEY(0,K,S):: IF S=0 THEN 170
180 M=7-M !ALTERNATE BETWEEN 3 & 4
190 FOR DELAY=1 TO 50 :: NEXT DELAY
200 GOTO 160
```

ter will be four times as large as the normal character and be in a 2×2. The color of the four sprites used together is the same. Also, if you magnify one sprite, you magnify all of them. You cannot have small and large on the screen at the same time.

Listing 18-5A

Line 130 clears the screen and sets the variable M to three. This variable will specify which magnification the computer should use.

Line 140 creates four new characters. The new character will begin with ASCII 128.

Line 150 uses the CALL SPRITE command to place the duck (Fig. 18-6) on the screen.

Line 160 places the message on the screen that tells the user which magnification the computer has

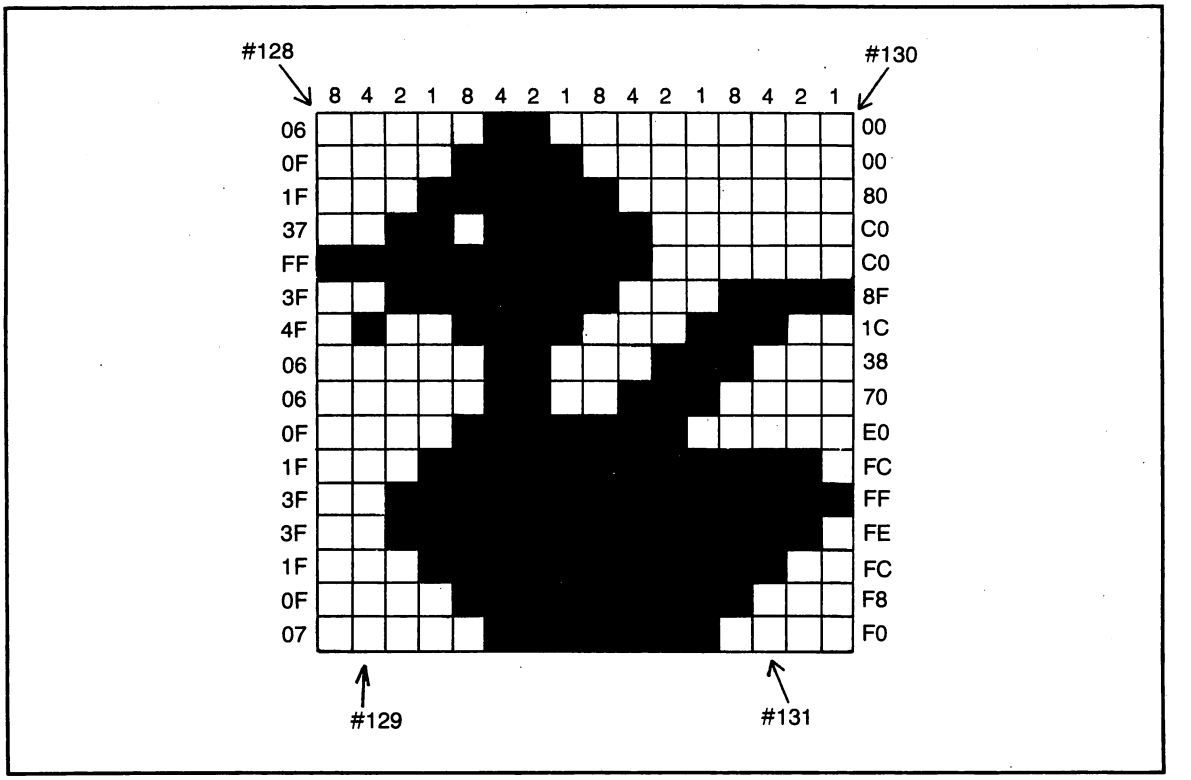


Fig. 18-6. Character for Listing 18-5A The Ducks.

used for the sprite.

Line 170 uses the CALL KEY command to wait for a key to be pressed. The computer will loop at this line until a key has been pressed to abort the program.

Line 180 subtracts the value of M from 7. This will alternate the value of M between 4 and 3.

Line 190 is a delay loop. Without it, the computer would shift the sprite between the three and four too fast.

Line 200 sends the computer back to line 160 for another key.

Even though you can only have four sprites on the screen at one time, if you use the 2×2 character combination for sprites, each character that makes up the sprite is not counted separately. The two characters that are in the same row for the sprite are counted as one sprite (Fig. 18-7). This is handy if you want to display words in large letters and the word consists of five to eight letters. In the program in Listing 18-5B, we use sprites to print large colorful letters on the screen.

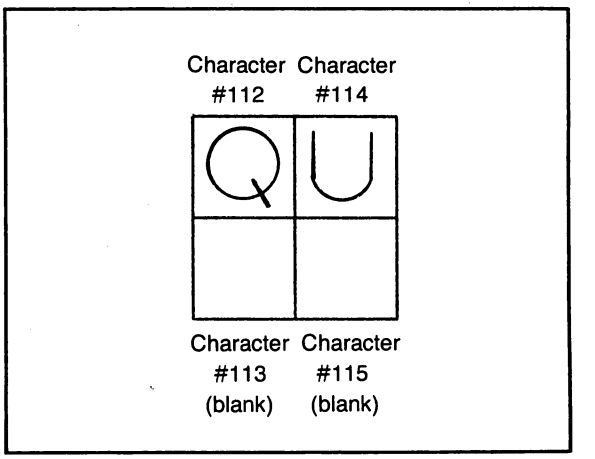


Fig. 18-7. Arrangement of Sprites for Magnify (3) and Magnify (4).

Listing 18-5B

```
100 REM LISTING 18-5B
110 REM THE DUCKS
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CLEAR :: M=3 :: MESSAGE$="QUACK
ERS"
140 CODE=112 :: FOR C=1 TO 8 :: CALL CHA
RPAT(ASC(SEG$(MESSAGE$,C,1)),TEMP$):: CA
LL CHAR(CODE,TEMP$):: CODE=CODE+1
150 CALL CHAR(CODE,"00"):: CODE=CODE+1 :
: NEXT C
160 CALL CHAR(128,"060F1F37FF3F4F06060F1
F3F3F1F0F0700008OC0C08F1C3870E0FCFFEFCF
8F0")
170 CALL SPRITE(#1,128,5,100,50,#2,128,3
,100,100,#3,128,7,100,150,#4,128,2,100,2
00)
180 CALL SPRITE(#5,112,5,132,104,#6,116,
3,132,120,#7,120,7,132,136,#8,124,2,132,
152)
190 DISPLAY AT(10,8):"MAGNIFY AT";M :: I
F M=4 THEN CALL LOCATE(#5,148,82,#6,148,
114,#7,148,146,#8,148,178)
200 IF M=3 THEN CALL LOCATE(#5,132,104,#
6,132,120,#7,132,136,#8,132,152)
210 CALL MAGNIFY(M)
220 CALL KEY(0,K,S):: IF S=0 THEN 220
230 M=7-M ! ALTERNATE BETWEEN 3 & 4
240 FOR DELAY=1 TO 50 :: NEXT DELAY
250 GOTO 190
```

Listing 18-5B

Line 130 clears the screen, sets the variable M to three for the magnification of the sprite and places the word "quackers" in the string. The letters in this word will be used as sprites.

Line 140 sets the variable CODE to 112. This is the first character that will be created. The FOR . . . NEXT loop places the character pattern for each letter of the message into TEMP$. This character pattern is then transferred to the new location pointed to by CODE. The variable CODE is incremented by one for the next location.

Line 150 places a set of zeroes in the next location and adds one to the value of CODE. The loop continues until all the letters from the message have been moved to the new location. Every other character will be a blank. When the word will be formed using the 2×2 sprite, we do not want any letters or characters under the words, so these characters must be blanks.

Line 160 places the four characters for the duck in the correct character locations.

Line 170 sets the first four sprites to character 128. Each duck will be a different color and in a different column, but they will all be in the same row.

Listing 18-6

```
100 REM LISTING 18-6
110 REM FLIGHT
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CHAR(128,"000042A518",129,"0081
422418",130,"0000000245A81")
140 CALL CLEAR :: CALL SCREEN(6)
150 CALL SPRITE(#1,128,16,32,192,0,-3)
160 FOR M=0 TO 2 :: CALL PATTERN(#1,128+
M)::: FOR DELAY=1 TO 50 :: NEXT DELAY ::
NEXT M
170 GOTO 160
```

Line 180 places the new characters for the letters in sprites five through eight. Each of these sprites will contain two letters and two blank characters.

Line 190 prints the message that tells the magnification of the sprites on the screen. The CALL LOCATE command is used to position the sprites that form the word on the screen.

Line 200 places the sprites at a different position if the magnification is a three.

Line 210 sets the magnification of the sprites.

Line 220 waits for a key to be pressed.

Line 230 subtracts the value of M from 7 to alternate the values between 3 and 4.

Line 240 delays the computer.

Line 250 sends the computer back to line 190 to wait for another key to be pressed.

CALL PATTERN

The CALL PATTERN command allows you to change to pattern of the character that the computer is using without affecting its color, speed, or location. The program in Listing 18-6 uses three different sprites for the bird (Fig. 18-8). As the bird flies across the screen, the computer changes the sprite and gives the illusion of actual flight.

Listing 18-6

Line 130 creates three new characters. These characters make up the bird's positions while it is flying.

Line 140 clears the screen and sets its color to blue.

Line 150 uses the character from location 128 as the

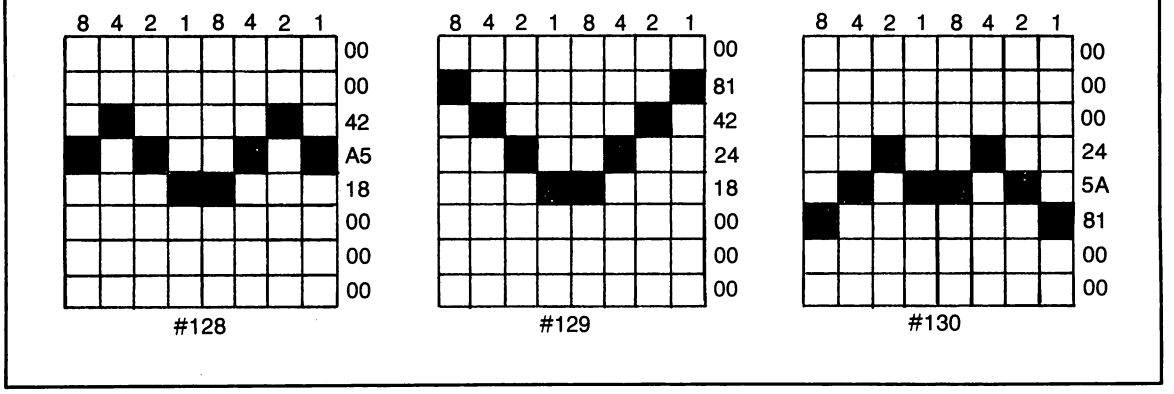


Fig. 18-8. Characters for Listing 18-6 Flight.

sprite. The sprite will move across the screen from right to left.

Line 160 is a FOR . . . NEXT loop. The CALL PATTERN command changes the pattern of the sprite from the character in location 128 to the character in location 129 and, finally, to the character in location 130. The delay loop slows the computer down so that the bird isn't flapping its wings too hard.

Line 170 sends the computer back to line 160. The speed of the bird will always remain constant as the computer changes the character that it places in that sprite.

CALL POSITION

This command returns the row and column of the sprite. You specify which sprite and the row and column will be placed in the variables.

```
CALL POSITION #1,ROW,COL
```

We are looking for the position of sprite number one. ROW will contain the row number of the sprite and COL will contain the column number.

CALL COINC

In many games, especially the arcade games, scores are based on the number of times one character, missile, or ball hits another or reaches its goal. You can constantly check the screen to see if the two characters or sprites are trying to occupy the same place on the screen, or you can use CALL COINC command. This command will place a negative one in the variable if the two sprites, or a sprite and character, are over each other. There are three different ways to use this command, as shown below:

```
CALL COINC(sprite #, sprite#, distance, variable)
CALL COINC(sprite#, row, column, distance, variable)
CALL COINC(ALL, variable)
```

In the first example, the two sprites that we are concerned with are identified by number. The distance in the number of minimum pixels that we will allow between the two sprites. The computer compares the upper left corners of the sprites to calculate the distance between them. The variable will be negative one if the two sprites are on top each other. The variable will be zero if the sprites are far enough apart. The second example compares the sprite to a fixed area on the screen. A hockey game might use this command to see if the puck hit the net or not. The third example compares all the sprites to each other and places a negative one in the variable if any two sprites occupy the same pixels on the screen.

CALL DELSPRITE

This command removes a sprite from the screen. The sprite must be recreated if you want to use it again.

```
CALL DELSPRITE(sprite #)
CALL DELSPRITE(ALL)
```

You can delete any or all sprites with this command. You can also specify more than one sprite in the program line by placing a comma between sprite numbers, as below:

```
CALL DELSPRITE(#1, #5)
```

CALL DISTANCE

The CALL DISTANCE command determines the distance between two sprites, or a sprite and a location on the screen. The distance is placed in a variable. This value is squared. You can use the squared value or the square root of the value depending on your program applications.

The program in Listing 18-7 illustrates several of the sprite commands in a simple arcade game. (See the flowchart in Fig. 18-9.)

Listing 18-7

Line 130 creates the characters that will be used for the ducks. These are the same characters that we used in the other duck programs.

Line 140 creates the characters that will be used for the arrow.

Listing 18-7

```
100 REM LISTING 18-7
110 REM SHOOTING GALLERY
120 REM BY L.M.SCHREIBER FOR TAB BOOKS
130 CALL CHAR(132,"060F1F37FF3F4F06060F1
F3F3F1F0F0700008OC0C08F1C3870E0FCFFFEFCF
8F0")
140 CALL CHAR(136,"0107190101010101010
101010103058OE0988080808080808080808080C
0A0")
150 CALL CLEAR :: CALL SCREEN(11):: CALL
 MAGNIFY(3):: D=4 :: SCORE=100
160 CALL SPRITE(#2,132,5,32,73,#3,132,3,
32,137,#4,132,7,32,201,#5,132,2,32,256)
170 DISPLAY AT(2,17):"SCORE:";SCORE
180 COL=128 :: CALL SPRITE(#1,136,13,176
,COL)
190 FOR SP=2 TO 5 :: CALL MOTION(#SP,0,-
10):: NEXT SP
200 CALL KEY(1,K,S):: IF S=0 THEN 200
210 IF K=2 THEN COL=COL-4
220 IF K=3 THEN COL=COL+4
230 IF COL<8 THEN COL=8 ELSE IF COL>236
THEN COL=236
240 CALL LOCATE(#1,176,COL)
250 IF K<>5 THEN 200
260 CALL MOTION(#1,-15,0)
270 FOR DUCK=2 TO 5
280 CALL COINC(#1,#DUCK,10,HIT):: IF HIT
 THEN 330
290 NEXT DUCK
300 CALL DISTANCE(#1,6,COL,F):: CALL SOU
ND(-800,110,30,110,30,F/256+500,15,-4,0)
310 CALL POSITION(#1,R,C):: IF R<6 OR R>
192 THEN CALL DELSPRITE(#1):: SCORE=SCOR
E-5 :: GOTO 170
320 GOTO 270
330 CALL COLOR(#DUCK,16):: CALL DELSPRIT
E(#1):: FOR C=1 TO 3 :: CALL SOUND(300,1
568,5-C,1568,5-C,1568,5-C):: NEXT C
340 FOR DELAY=1 TO 50 :: NEXT DELAY :: C
ALL DELSPRITE(#DUCK)
350 SCORE=SCORE+10 :: DISPLAY AT(2,23):S
CORE :: D=D-1 :: IF D THEN 180
```

```
360 DISPLAY AT(12,5):"PLAY AGAIN (Y/N) ?
" :: ACCEPT AT(12,24)BEEP VALIDATE("YN")
SIZE(1):A$ :: IF A$="" THEN 360
370 IF A$="Y" THEN 150
380 CALL CLEAR
```

Line 150 clears the screen, sets the screen color and sets the sprite mode to the MAGNIFY(3). This is the four-character sprite in the normal mode. The D variable will keep track of how many ducks are on the screen and the score begins with 100 points.

Line 160 places the duck characters into sprite numbers two through five. Each duck will be a different color, and be in a different column. They will all be in the same row on the screen.

Line 170 places the score on the screen.

Line 180 sets the variable COL to 128. This is the column that the arrow will be printed at. Sprite number one will be the arrow.

Line 190 uses a FOR . . . NEXT loop to make the other sprites move. The four duck sprites will move across the screen horizontally at the same speed.

Line 200 checks to see if a key has been pressed. The computer will only consider the keys on the left side of the keyboard. The computer will loop at this line until a key is pressed.

Line 210 compares the value of K with two. If the S key has been pressed, the COL variable will have four subtracted from it.

Line 220 checks to see if the D key has been pressed. When this key is pressed, four is added to the value of COL. This moves the arrow to the left or right on the screen.

Line 230 checks the value of COL to make sure that it will not be printed off the screen. If the value of COL is too large or too small, it will be reset to the screen edge position.

Line 240 uses the CALL LOCATE command to place the arrow on the screen. The arrow will be erased from its old position and placed in its new position.

Line 250 checks to see if the E key was pressed. This key is the trigger key. It releases the arrow and allows it to move up the screen at the row of ducks. If the E key has not been pressed, the computer will go back to line 200 and wait for another key to be pressed.

Line 260 sets the first sprite in motion. The arrow will travel up the screen in a straight line slightly faster than the ducks are moving across the screen.

Line 270 begins the FOR . . . NEXT loop that checks to see if a duck has been hit. The loop begins with 2 and ends with 5 because these are the sprite numbers of the ducks.

Line 280 uses the CALL COINC command to see if the arrow is hitting the duck specified by the variable DUCK. The 10 is the maximum distance that the arrow can be from the duck to consider it a hit. If the arrow has hit the duck, the variable HIT will be set to negative one. If it is negative one, the computer will go on to line 330 to remove the duck from the screen.

Line 290 continues the loop until all four ducks are checked for a hit.

Line 300 uses the CALL DISTANCE command to see how far the arrow is from the top of the screen. The distance is stored in variable F. This value is used in the CALL SOUND command to make a sound based on the distance of the arrow from the top of the screen.

Line 310 uses the CALL POSITION command to see if the arrow sprite has reached the top of the screen. If it has, the CALL DELSPRITE is used to remove the arrow from the screen. The player missed the duck. The score is decreased by five and the computer goes to line 170 to update the score on the screen. The player can try again.

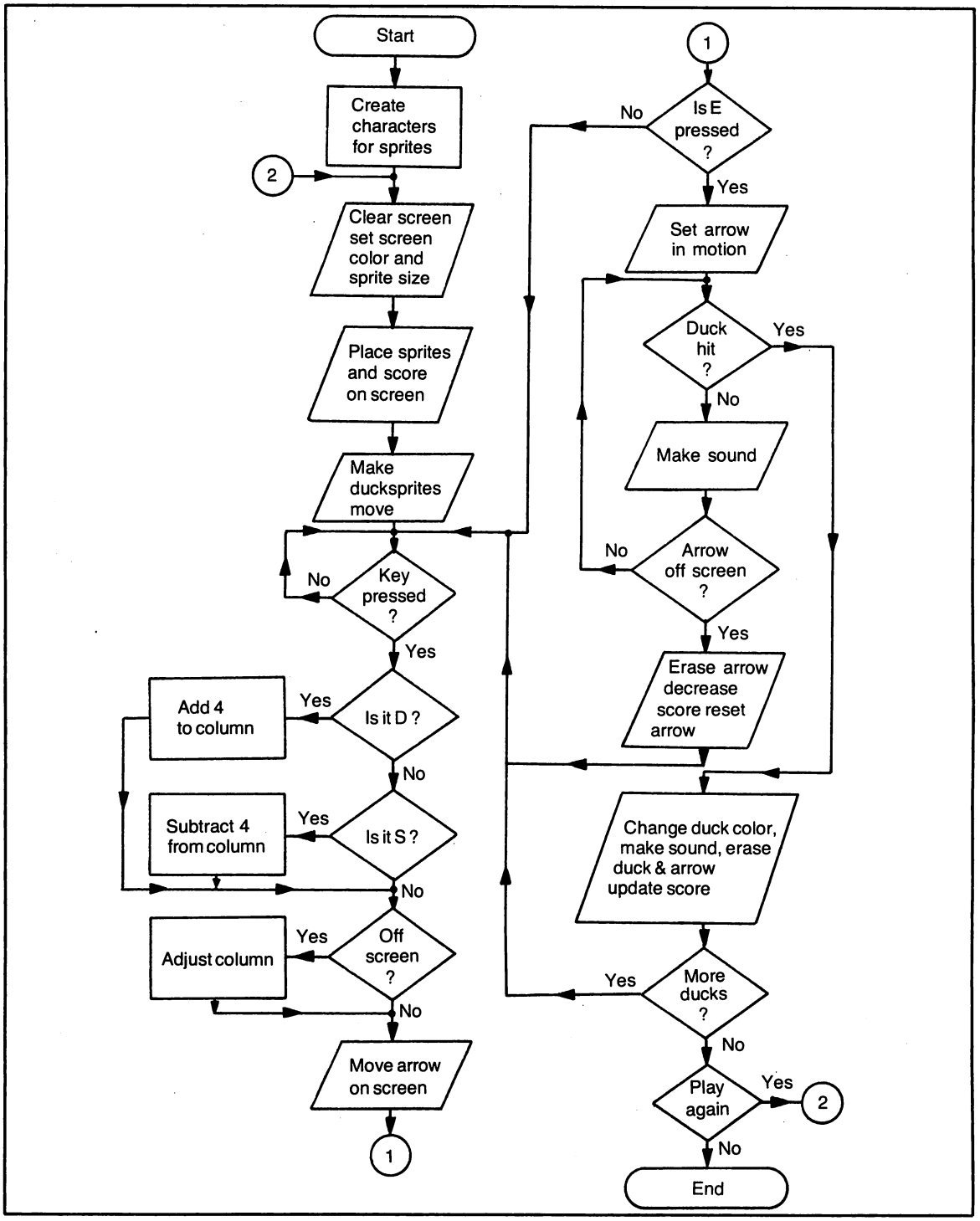


Fig. 18-9. Flowchart for Listing 18-7 Shooting Gallery.

Line 320 sends the computer back to line 270 where it will check once again for a hit on the ducks. The computer will continue to loop at these lines until a duck has been hit or the arrow travels off the top of the screen.

Line 330 begins the routine for the hit duck. First, the color of the duck is changed with the CALL COLOR command. Any one sprite color can be changed with this command. Next, the arrow sprite is removed from the screen with the CALL DELSPRITE command. The FOR . . . NEXT loop makes a bell sound indicating that a duck has been hit.

Line 340 is a delay loop to slow down the program. Then the duck is removed from the screen.

Line 350 adds ten points to the score and updates the score on the screen. One duck is subtracted from the D variable. If there are ducks on the screen, the computer will go back to line 180 for another turn.

Line 360 asks if you would like to play again. If you do not enter any letter, the computer will loop until you do.

Line 370 sends you to line 150 for a new game if you enter a "Y."

Line 380 clears the screen. The program ends because you entered an "N."

Chapter 19
Using The Disk

Sooner or later you'll find the cassette is too slow for you, or your programs need the random-access capabilities of the disk. This chapter will give you an overview of the commands that are available with the disk drive. When adding a disk drive to your TI-99/4A, you should also add the Memory Expansion Unit. Your disk drive, extra memory, and other accessories can be added through this unit.

In addition to saving programs to and loading programs from the disk, you can access the disk from a BASIC program.

ACCESSING THE DISK

OPEN/CLOSE

The OPEN command must be used before you can access the disk. It opens a file through which you can input, output, update, or append the files on the disk. We used the OPEN command in Chapter 14 when we saved the new character set to the disk. The format for this command is:

OPEN #file:device{options}

The file number must be a number between 1 and 255. File zero is used by the keyboard and should not be used for the disk. The device is the disk or cassette. To indicate the disk, use DSK1, followed by the file name. The options should be specified in this order:

FILE-ORGANIZATION—The files can be stored on the disk as random files or sequential files. A random file means the records or information in that file can be read or written in any order. In a sequential file the records or information must be accessed in order beginning with the first record. If the entire file will be loaded from the disk at one time and used within the program, the sequential files should be used. If you will be using the information from the disk while it is on the disk, and you will be accessing different information from different areas of the file, then random files should be specified. The default for the file organization is sequential.

FILE-TYPE—is either display or internal. When you specify display, the information is stored in ASCII code. The internal type is binary code. If you store your records in binary, they will take up less room on the disk and be saved and/or loaded more quickly. The default is display.

OPEN-MODE—is the option that tells the computer whether it is reading or writing to the disk. You can specify update, input, output, or append. With update, you can read and write to the disk. Input will only read from a disk and output limits you to writing to the disk. The append will only add one file. The default is update.

RECORD-TYPE—tells the computer whether the records that will be storing or retrieving from the disk are all the same length or different lengths. Specify that they are variable if all the records may not be the same length. Specify fixed if they will all be the same length. If you are using the relative record-type, you must use the fixed record type. You can also specify how long the records will be for either the fixed or variable record-types.

Once you have stored or retrieved the information, you will use the CLOSE command to close the file. The file that will be closed is the same file number as the one that you were using: CLOSE #1. Once you close a file, the computer cannot access it without another OPEN command. It is good programming practice to close files once you have used them to prevent accidental damage, (changes in or erasure of information) to open the file.

STORING TO AND RETRIEVING FROM DISK

PRINT

Just as the PRINT command places information on the screen, you can use it with the files to print information to the disk. The command must be followed by the file number. This file has already been opened with the open command. If the file is relative, you can also specify the record number that you want the information placed in. Use print in the format below:

PRINT #file{,REC number}: information

INPUT

Once you have the information on the disk, you will want to be able to bring it back into the computer. The INPUT command followed by the file number will allow you to read the information from the disk.

INPUT #file{,REC number}: variable

The file must be opened and the same file number used. If the file is relative, and you want to read the information from a specific record, you can specify which record number you want to read. The variables, string or numeric, follow the colon. They will contain the information that is read from the disk.

LINPUT

This command is similar to the INPUT command, but it can be much quicker when working with large files. Instead of bringing in just one piece of information at a time, the LINPUT command brings in the entire record and stores it in a string. Then, your program can use the information in the string. If the file is relative, you can specify which record you want the computer to read.

LINPUT #file{,REC number}: string variable

REC

This command will tell you the number of the next record that will be read or written. It is useful if the program that you are writing needs to know what record number is next. The format is REC(file-number). The file number is the file that is opened. You do not need the number sign before the number.

EOF

The letters EOF mean end-of-file. This command is very useful if you are reading a file and you do not know how many records have been stored. With an IF . . . THEN statement you can direct the computer to read another record or close the file.

```
100  IF EOF(1)= THEN 200 ELSE 80
```

This program statement tells the computer that if the end-of-file value for file one is a one, then go to line 200. If it is not, go to line 80 to read the next record. The EOF value will be a one; if you are at the last record of the file. It will be zero if there are more records for this file. It will be a negative one if there is no more room on the disk. The number in the parentheses is the file number that has been opened.

MERGING DISK FILES

MERGE

This command has been used in previous chapters. It is a very easy to combine frequently used program routines without having to retype the entire routine. The MERGE command will bring in a program from disk and add it to the program in the computer. If any line numbers in the program on disk are the same as the line numbers of the program in the computer, the lines from the program on disk will replace the lines in the computer.

All programs or routines cannot be merged with another. In order for the merge to work, the program or routine must be saved to the disk with the merge option below:

```
SAVE DSK1.name,MERGE
```

To add the program to the program in the computer type:

```
MERGE DSK1.name
```

Now you can store your favorite routines on disk and bring them in when you write your next program. One tip here, make these routines subroutines with high line numbers that you would not use in your program. After you bring in the routines from disk, you can renumber your program.

DELETING DISK FILES

DELETE

You will not always want to keep all the programs that you have saved on your disk. With the cassette, you can just record over a program. It doesn't work that way with a disk. You have to tell the computer to remove a program from the disk. After a program is removed, the computer can reuse that disk space for another program. You can delete a file using a direct command or from a BASIC program. The format is:

```
DELETE "DSK1.filename"
```

Chapter 20

Putting It All Together—Using Sprites, Special Characters, And Sound

Now that you can redefine characters, use sprites, make music and save information to cassette or disk, you are ready to develop programs that use all these features. The last program in this book (Listing 20-1) is a classical puzzle—the Towers of Hanoi. There are nine disks on the first pole. The object of the puzzle is to move the nine disks, one at a time, from the first pole onto the third pole. A smaller disk can be placed on a larger disk, but a larger disk can never be placed on a smaller one. The magnet is a sprite. You use the joystick to move the magnet over one of the three poles and press the fire button to pick up or drop a disk. Move the joystick to the left or right to move the magnet, with or without a disk. The program will not allow you to drop a larger disk on a smaller one. Each time you move a disk, even if you pick it up and drop it back onto the same pole, the number of moves is increased by one.

Listing 20-1

Line 130 sets aside the memory for the array that will keep track of which disks are on which poles and in what order.

Line 140 clears the screen and creates the characters that will be used in the program.

Lines 150-180 place each disk in an element of R$ array.

Lines 190-230 contain the character patterns.

Line 240 changes certain characters in the character set. The C variable is the character that will be changed; C$ is the character pattern for the redefined character.

Lines 250-260 set the screen color, change the colors of the character sets, and set the mode of the sprite to MAGNIFY, (2).

Lines 270-280 places the title of the program (TOWERS), the characters, and the number of moves on the screen. The MOVE variable will increase by one every time a disk is picked up and dropped.

Line 290 places the three poles and the platform on the screen.

Line 300 places the tops on the poles.

Line 310 uses the CALL SPRITE command to make

Listing 20-1

```
100 REM LISTING 20-1
110 REM TOWERS GAME
120 REM BY A.R.SCHREIBER FOR TAB BOOKS
130 DIM P(9,3)
140 CALL CLEAR :: FOR CNT=0 TO 6 :: READ
 C$ :: CALL CHAR(120+CNT,C$,128+CNT,C$,1
36+CNT,C$):: NEXT CNT
150 R$(1)=CHR$(121)&CHR$(120)&CHR$(122):
: R$(2)=CHR$(131)&CHR$(128)&CHR$(132)::
R$(3)=CHR$(141)&CHR$(136)&CHR$(142)
160 R$(4)=CHR$(120)&CHR$(120)&CHR$(120):
: R$(5)=CHR$(129)&CHR$(128)&CHR$(128)&CH
R$(128)&CHR$(130)
170 R$(6)=CHR$(139)&CHR$(136)&CHR$(136)&
CHR$(136)&CHR$(140):: R$(7)=CHR$(125)&CH
R$(120)&CHR$(120)&CHR$(120)&CHR$(126)
180 R$(8)=CHR$(128)&CHR$(128)&CHR$(128)&
CHR$(128)&CHR$(128):: R$(9)=CHR$(137)&CH
R$(136)&CHR$(136)&CHR$(136)&CHR$(136)&CH
R$(136)&CHR$(138)
190 DATA FFFFFFFFFFFFFFFF,0303030303030303
03,COCOCOCOCOCOCOCO,OFOFOFOFOFOFOFOF,FOF
OFOFOFOFOFOFO,3F3F3F3F3F3F3F3F
200 DATA FCFCFCFCFCFCFCFC,112,3C66C3C3C3
C3C3C3,113,FFFFFFFFFFFFFFFF,114,3C7EFFFF
FFFFFFFF,111,FFFFFFFFFFFFFFFF
210 DATA 115,3C42A02222242830,116,D01010
10080E,96,00007C7C10101010,97,1010101010
101010,98,0000387C44444444,99,4444444444
447C38
220 DATA 100,0000444444444444,101,747474
7474742828,102,00007C7C40404040,103,7878
404040407C7C
230 DATA 104,0000787844444444,105,787850
5048484444,106,0000383844444040,107,3838
040444443838
240 FOR CNT=1 TO 18 :: READ C,C$ :: CALL
 CHAR(C,C$):: NEXT CNT
250 CALL SCREEN(2):: CALL COLOR(3,16,1,4
,16,1,5,4,1,6,4,1,7,4,1,9,14,1,10,14,1,1
1,16,1):: CALL MAGNIFY(2)
260 CALL COLOR(12,3,1,13,11,1,14,7,1)
270 DISPLAY AT(1,10):"s `bdfhj s"
```

```
280 DISPLAY AT(2,10):"t ac@sik t" :: DIS
PLAY AT(4,4):"MOVES";TAB(10);MOVE
290 CALL VCHAR(12,7,113,12):: CALL VCHAR
(12,17,113,12):: CALL VCHAR(12,27,113,12
):: CALL HCHAR(24,1,111,32)
300 DISPLAY AT(11,5):"r          r
  r"
310 CALL SPRITE(#1,112,16,41,45)
320 FOR CNT=1 TO 9 :: L=LEN(R$(CNT)):: D
ISPLAY AT(14+CNT,5-L/2)SIZE(L):R$(CNT)::
 NEXT CNT
330 FOR CNT=1 TO 9 :: P(CNT,1)=CNT :: NE
XT CNT :: MP=45 :: ML,RL,ORP=0 :: RP=5 :
: POLE=1
340 CALL JOYST(1,X,Y):: IF SGN(X)=0 THEN
 CALL KEY(1,K,S):: IF S=0 THEN 340
350 IF SGN(X)=-1 AND MP<>45 THEN MP=MP-8
0 :: RP=RP-10 :: POLE=POLE-1
360 IF SGN(X)=1 AND MP<>205 THEN MP=MP+8
0 :: RP=RP+10 :: POLE=POLE+1
370 IF SGN(X)<>0 THEN CALL LOCATE(#1,41,
MP):: DISPLAY AT(8,ORP-(RL/2)):"" :: DIS
PLAY AT(8,RP-(RL/2)):R$(ML):: GOTO 340
380 REM PICK UP OR DROP A RING
390 IF ML<>0 THEN 500
400 REM PICK UP A RING
410 FOR CNT=1 TO 9
420 ML=P(CNT,POLE):: IF ML<>0 THEN P(CNT
,POLE)=0 :: GOTO 440
430 NEXT CNT :: GOTO 340
440 RR=14+CNT :: RL=LEN(R$(ML)):: FOR CN
T=RR TO 9 STEP -1
450 IF CNT>11 THEN DISPLAY AT(CNT,RP-3)S
IZE(7):"   q   " ELSE IF CNT=11 THEN DIS
PLAY AT(CNT,RP-3)SIZE(7):"   r   "
460 IF CNT<11 THEN DISPLAY AT(CNT,RP-(RL
/2))SIZE(RL):""
470 DISPLAY AT(CNT-1,RP-(RL/2))SIZE(RL):
R$(ML)
480 NEXT CNT :: CALL SOUND(50,1760,0)::
GOTO 340
490 REM DROP A RING
500 FOR CNT=1 TO 9
```

```
510 TR=P(CNT,POLE)!! IF TR<>0 THEN 540
520 NEXT CNT
530 REM MAKE SURE TOP RING,IF ANY, IS LA
RGER THAN MAGNET'S RING
540 CNT=CNT-1 !! IF TR<ML AND TR<>0 THEN
 340
550 P(CNT,POLE)=ML !! RR=13+CNT !! RL=LE
N(R$(ML))!! FOR CNT=8 TO RR
560 IF CNT<11 THEN DISPLAY AT(CNT,RP-3)S
IZE(7)!"" ELSE IF CNT=11 THEN DISPLAY AT
(CNT,RP-3)SIZE(7)!"    r    "
570 IF CNT>11 THEN DISPLAY AT(CNT,RP-3)S
IZE(7)!"    Q    "
580 DISPLAY AT(CNT+1,RP-(RL/2))SIZE(RL)!
R$(ML)
590 NEXT CNT !! CALL SOUND(50,440,0)!! M
OVE=MOVE+1 !! DISPLAY AT(4,10)!MOVE !! M
L=0
600 IF P(1,3)<>1 THEN 340
610 REM THE TASK WAS ACCOMPLISHED
620 FOR COLR=3 TO 16 !! FOR CNT=8 TO 14
!! CALL COLOR(CNT,COLR,1)!! COLR=COLR+1
!! IF COLR>15 THEN COLR=3
630 CALL SOUND(50,COLR*110,0,COLR*220,0,
COLR*330,0)
640 CALL KEY(0,K,S)!! IF S<>0 THEN 660
650 NEXT CNT !! NEXT COLR
660 CALL CLEAR
```

the first sprite the magnet.

Line 320 places the disks on the first pole. The row that the disk is printed on is the value of CNT offset by 14. The disks will be printed from the smallest one down to the largest. On the screen, the smallest one is on the top and the largest on the bottom.

Line 330 places the value of CNT into the first element of the array. The array will contain values from one to nine. Each value represents that disk, with one the smallest disk and nine the largest. The first set of elements of the array represent the first pole. The array is 9×3. The second and third set are zeroes because there are no rings on those poles. The other variables in this line keep track of which pole the magnet is over, the magnet's current position, and the position of the rings.

Line 340 checks to see if the joystick has been moved. We are only interested in seeing if the joystick is moved to the right or left. If the sign of the X is zero, the joystick has not been moved and the computer continues with the commands on this line. The CALL KEY command is used to see if the fire button has been pressed on the joystick. If it has not, the computer loops back to the beginning of this line and waits for the joystick to move or for the fire button to be pressed.

Line 350 checks to see if the sign of X is negative. If it is, the joystick has been moved to the left. Now the computer checks to see if the magnet can move to the left. The variable MP cannot be 45. If it is, the magnet cannot move to the left. If it isn't, 80 is subtracted from MP. This is the new position for the magnet. The ring will also have a new position, and the pole that the magnet will be over is one less than the pole it is currently over.

Line 360 checks to see if the sign of the variable X is positive. If it is, then the joystick was moved to the right. The computer checks the value of the variable MP. If it is not 205 the magnet can move to the right. The next position of the magnet is 80 pixels to the right. The ring and pole positions are also changed.

Line 370 verifies that the joystick has been moved. If the sign of X is zero, the joystick has not moved. If it is not zero, the joystick has moved and the computer will move the sprite to the new position using CALL LOCATE. If a ring is under the magnet, it will also move. The computer is directed back to line 340 again to see if the joystick has been moved or the fire button pressed.

Line 390 checks the value of ML. If it is not zero, there is a ring under the magnet, and the computer is directed to the line to drop the ring.

Line 410 begins a FOR . . . NEXT loop. This loop will check every position under the magnet to see if there is a ring under the magnet. If there is, the computer will be able to pick it up and move it.

Line 420 checks the positions in the array as indicated by the pole number. If the array element does not contain a zero, there is a ring there. Since we are removing the ring from that position, a zero is placed in that array element.

Line 430 continues the loop until there are no more elements in the array. If the computer looks at every element of the array and does not find a ring on the pole, the computer will be sent back to line 340. The magnet must be moved to another pole.

Lines 440-480 pick up the ring. The value of CNT is offset by 14 so the computer will move the ring in the correct row on the screen. The lower case "Q" replaces the ring. This character is the pole. The ring is printed one row higher on the screen. This action continues until the ring is in the row under the magnet. CALL SOUND is used to make a metallic sound and the computer is sent to line 340.

Line 500 begins the routine to drop the ring. The FOR . . . NEXT loop checks the elements of the array under the pole. The ring cannot be dropped onto a smaller ring.

Line 510 checks the element for a zero. If it is a zero, the loop continues. If there is a ring in this position, the computer is sent to line 540 to check its size.

Line 520 continues the FOR . . . NEXT loop. If all the elements are checked and none contain a ring, the program continues, because the ring can be dropped on an empty pole.

Line 540 subtracts one from the value of CNT. This is the position in the array and on the pole that the ring will occupy. If the value of the ring on the pole is less than the value of the ring on the magnet, the computer goes back to line 340 and the ring will not drop.

Lines 550-590 drop the ring onto the pole. The lower case R is the top of the pole, the lower case Q is the pole. The ring is printed on the screen, then the pole is printed over it and the ring is printed one position lower. After the ring reaches the bottom of the pole, the CALL SOUND is used to make the thump sound. The counter for the number of moves is incremented by one.

Line 600 checks the array element for the first position of the third pole. If it contains a one, then the task was accomplished and the computer can go on to the ending routine. If it has not been accomplished, the computer will be sent to line 340 and wait for another move.

Lines 620-650 contain nested FOR . . . NEXT loops. These loops continually change the values of the colors in the character sets. The sound made is based on the colors on the screen. The CALL KEY is used to see if a key has been pressed. This fanfare will continue until a key has been pressed.

Line 660 clears the screen and ends the program.

Appendix

Working with Numbers

Ever since man had the need to know how many items he had in his possession, how much grain he needed, or how many days since the last rain, he had to devise a system to count. It is believed that some ancient tribes used the base two or three for counting. There is some evidence that base twenty was used by a few early tribes, since their handiest counting aids were their fingers and toes.

With numbers came the need to do simple calculations. Soon the problems were no longer simple, and man quickly learned that if he marked the numbers in the dirt, or on a tablet, he could compute much more quickly. Stones were probably used much the same way we use poker chips today with each type of stone representing a different group of numbers—ones, fives, tens, and the like.

The abacus is the oldest, yet the simplest, adding machine invented. The principle of moving the beads on rods has survived the test of time. Many people consider the abacus the first type of computer.

THE BINARY SYSTEM

As with the abacus, the computer uses its own number system—binary. If you think of a light bulb, a lock, or a trap, each item has only two states. It can be either on or off, open or closed, set or sprung. The computer operates in the same manner. Each memory location in the computer can be either on or off.

The memory in your computer can hold a charge. This is represented by the number one. When a location has no charge, it is represented by a zero. The computer, then, uses binary or base two for its number system.

In our decimal system, each number position is a multiple of 10. The position before the decimal is the unit position. In the binary system, each position is a multiple of two with the position before the decimal the unit position. In the decimal system, there are 10 numerals, 0 through 9. In the binary system, only the numbers 0 and 1 are used. The binary number 10110 is 22 in decimal. To

convert a binary number to decimal, we add the places that contain a 1 and ignore the place values where there is a zero.

128	64	32	16	8	4	2	1
0	0	0	1	0	1	1	0

In our example, 10110, there is 1 in the 16's columns, a 1 in the 4's column and a 2 in the 2's column. If we add 16 + 4 +, we arrive at 22 or the decimal equivalent. Most computers have 8 positions in each memory location. This means that each location can contain a number from 0 to 255.

The number that is stored in each memory location is called a byte. Each one or zero in the byte is referred to as a bit. Although the TI-99/4A computer is considered a 16-bit computer, some of the memory is treated as eight-bit. There are some four-bit and eight-bit computers also. Each byte can also be divided into two four-bit nibbles.

Although it seems confusing at first, using the binary system in computers conserves on parts and increases speed. If a switch with ten different settings were used, the computer would have to determine if the switch was set, then decide which setting it was pointed to. In binary, there are only two possibilities, a 1 or 0. It takes only eight bits (switches) to count to 255. By adding eight more, any number up to 65535 can be displayed. Work the following examples to practice converting binary numbers to decimal.

1. 01100001
2. 10110111
3. 11001000
4. 00111001
5. 01110010
6. 00111100
7. 00011110
8. 11011000
9. 01111010
10. 11110001

The decimal equivalents are: 1-97; 2-183; 3-200; 4-57; 5-114; 6-58; 7-30; 8-216; 9-122; 10-241

UNDERSTANDING HEX

Although the binary system increases the computer's speed, most of us cannot readily convert a string of ones and zeros into a number that we can understand. To help us, most programmers and manuals reference the memory locations and the number stored in it in hex. The hexadecimal system uses base 16. The numbers after nine are represented as the letters A-F. To convert a binary number to hex, we first divide the byte into two nibbles. If, for example, we needed to convert 11001101 into hex, we would divide it into two nibbles—1100 1101. Each nibble consists of four bits. Now we treat each nibble as a separate number. By adding the place values of the first nibbles, 8 + 4, we get 12. Twelve is not a one-digit number, so we use the letter "C." The next nibble is 8 + 4 + 1, or 13. One number more than "C" is "D." Our hex number for 11001101 is CD.

Let's try that again with another binary number, 10010111. Divide this eight-bit number into two nibbles—1001 0111. The first nibble is 8 + 1 or 9, the second is 4 + 2 + 1 or 7. The hex number for 10010111 is "97."

There are times when you want the decimal equivalent to a hex number. When you are working in BASIC and want to POKE a location with a number, both the location and the number that you are POKEing must be in decimal. Often the manual you are using will only provide the HEX addresses to be POKEd or the HEX values that should be entered. To convert a HEX number to decimal is fairly easy. Since each number/letter represents a value from one to 15, each place value in HEX is a multiple of 16. If the HEX number has only two places such as B3, multiply the number in the second position from the end by 16 and add the value in the rightmost position. B is equal to 11 decimal, 11 × 16 is 176. Add 3 and the decimal value of HEX B3 is 179. Since the computer can access over 64000 memory locations, often the HEX number will contain four places. To convert C253 HEX to decimal we would multiply the "C" (decimal 12) by

4096, the 2 by 256, the 5 by 16 and add 3.f
(12×4096)+(2×256)+(5×16)+3=49747

To convert a decimal number to HEX, divide the number by the largest place value feasible, the quotient is the value for that place, divide the remainder by the next place value, and continue until there is a remainder less than 16. That remainder is the last digit of the HEX number. If, for example, the decimal number is 21013, we would divide the number by 4096. The first or leftmost value of the HEX number is a 5. The remainder is 533. When this number is divided by 256, the next quotient is a 2 with a remainder of 21. 21 divided by 16 is 1 with a remainder of 5. Therefore, the HEX equivalent of 21013 is 5215, as shown below:

```
      5          2          1
4096/21013   256/533    16/21
-20480       -512       -16
  533          21         5
```

The program in Listing A-1 will convert a decimal number to Hex or binary and a binary or HEX number to decimal.

Listing A-1

Line 130 traps for errors that would cause a warning message to occur.
Lines 140-160 place the menu on the screen.
Line 170 prints a question mark and waits for a selection.
Line 180 uses the ON GOSUB command to send the computer to the right based on the selection.
Line 190 sends the computer back to the beginning of the menu when the computer returns from the subroutine.
Lines 210-230 print the message for the first routine. This routine will convert a decimal number into a hexadecimal number.
Line 240 clears A$. If we did not clear the string each time we began this routine, the hex digits would be added to the digits in A$. The conver-

Listing A-1

```
110 REM CONVERSIONS
120 REM BY L.M. SCHREIBER FOR TAB BOOKS
130 ON WARNING NEXT
140 DISPLAY AT(3,1)ERASE ALL:"PLEASE SEL
ECT A CONVERSION"
150 DISPLAY AT(5,2):"1. DECIMAL TO HEX"
:: DISPLAY AT(7,2):"2. DECIMAL TO BINARY
"
160 DISPLAY AT(9,2):"3. HEX TO DECIMAL"
:: DISPLAY AT(11,2):"4. BINARY TO DECIMA
L"
170 DISPLAY AT(14,8):"?" :: ACCEPT AT(14
,9)VALIDATE("1234")SIZE(1):N
180 ON N GOSUB 210,310,390,480
190 GOTO 140
200 REM ROUTINE TO CONVERT DECIMAL NUMBE
RS TO HEX
210 DISPLAY AT(1,1)ERASE ALL:"PLEASE ENT
ER THE DECIMAL     NUMBER TO BE CONVERTED
 TO"
```

```
220 DISPLAY AT(3,1):"HEX.  NUMBER CANNOT
 BE       GREATER THAN 65535."
230 DISPLAY AT(7,1):"TO EXIT, ENTER (0)
ZERO"
240 A$="" :: ACCEPT AT(9,7)VALIDATE(DIGI
T):DGIT
250 IF DGIT=0 THEN RETURN
260 IF DGIT>65535 OR DGIT<0 THEN 240
270 STDGIT=DGIT ! STORE THE NUMBER FOR T
HE CONVERSION ROUTINE
280 V=65536 :: FOR HEXP=1 TO 3 :: V=V/16
 :: GOSUB 570 :: NEXT HEXP
290 V=V/16 :: H=STDGIT :: GOSUB 600
300 DISPLAY AT(3,1)ERASE ALL:"THE HEX EQ
UIVALENT OF";DGIT;"IS ";A$ :: GOSUB 620
:: GOTO 210
310 DISPLAY AT(3,1)ERASE ALL:"PLEASE ENT
ER THE DECIMAL    NUMBER TO BE CONVERTED
 TO   BINARY."
320 DISPLAY AT(6,1):"NUMBER CANNOT EXCEE
D 255.   TO EXIT THIS ROUTINE ENTER 0 (Z
ERO)."
330 A$="" :: ACCEPT AT(9,9)VALIDATE(DIGI
T)SIZE(3):DGIT
340 IF DGIT=0 THEN RETURN
350 IF DGIT>255 THEN 330
360 STDGIT=DGIT ! STORE THE NUMBER ENTER
ED FOR THE CONVERSION ROUTINE
370 V=256 :: FOR PBIN=1 TO 7 :: V=V/2 ::
 GOSUB 570 :: NEXT PBIN :: A$=A$&STR$(ST
DGIT)
380 DISPLAY AT(3,1)ERASE ALL:"THE BINARY
 EQUIVALENT OF";DGIT;" IS  ";A$ :: GOSUB
 620 :: GOTO 310
390 DISPLAY AT(1,1)ERASE ALL:"PLEASE ENT
ER THE HEX NUMBER TO BE CONVERTED TO DEC
IMAL"
400 DISPLAY AT(3,1):"NUMBER CANNOT EXCEE
D FFFF   TO EXIT PRESS ENTER"
410 DGIT=0 :: ACCEPT AT(6,5)VALIDATE(DIG
IT,"ABCDEF")SIZE(4):A$
420 IF A$="" THEN RETURN
430 HEXP=LEN(A$)! FIND OUT HOW MANY POSI
TIONS
```

```
440 FOR V=1 TO HEXP :: C=ASC(SEG$(A$,V,1
))
450 C=C-55 :: IF C<10 THEN C=VAL(SEG$(A$
,V,1))!IF IT'S NOT A LETTER THEN GET THE
 VALUE
460 P=HEXP-V :: DGIT=DGIT+C*16^P :: NEXT
 V
470 DISPLAY AT(3,1)ERASE ALL:"THE DECIMA
L EQUIVALENT OF "&A$&" IS"&DGIT :: GOSUB
 620 :: GOTO 390
480 DISPLAY AT(1,1)ERASE ALL:"PLEASE ENT
ER THE BINARY      NUMBER TO BE CONVERTED
 TO    DECIMAL.  NUMBER CANNOT"
490 DISPLAY AT(4,1):"EXCEED 11111111.
 TO EXIT PRESS ENTER."
500 DGIT=0 :: ACCEPT AT(9,5)VALIDATE("10
")SIZE(8):A$
510 IF A$="" THEN RETURN
520 P=LEN(A$)
530 FOR V=1 TO P :: C=VAL(SEG$(A$,V,1))
540 IF C=1 THEN DGIT=DGIT+2^(P-V)
550 NEXT V
560 DISPLAY AT(3,1)ERASE ALL:"THE DECIMA
L EQUIVALENT OF "&A$&" IS "&DGIT :: GOSU
B 620 :: GOTO 480
570 IF STDGIT<V THEN A$=A$&"0" :: RETURN
580 H=INT(STDGIT/V)! DIVIDE THE NUMBER B
Y THE VALUE OF THE POSITION
590 STDGIT=STDGIT-H*V ! STORE THE REMAID
ER FOR THE NEXT CONVERSION
600 IF H>9 THEN A$=A$&CHR$(H+55):: RETUR
N
610 A$=A$&STR$(H):: RETURN
620 DISPLAY AT(10,5):"PRESS ANY KEY" ::
CALL KEY(0,K,S):: IF S=0 THEN 620 ELSE R
ETURN
```

sion would be correct only the first time that the routine was used. The number that you want to convert to hex is stored in the DGIT variable.

Line 250 checks to see if a zero was entered. The routine returns when a zero is entered.

Line 260 checks the number entered. If it is too large, or a negative number, the computer will be sent back to line 240 for another input.

Line 270 stores the number entered in STDGIT. The computer will use the number stored in this variable for the conversion routine.

Line 280 places the highest value that number can

be in the V variable. This number is divided by 16 in the FOR . . . NEXT loop to set the place values of the HEX number. The subroutine at line 570 will place the correct digit in A$.

Line 290 sets the digit for the last position in A$. This is the one's position, so the computer begins the subroutine at 600 since it has no conversion to make. It only needs to determine if the number will be numeric or a letter from A through F.

Line 300 places the conversion on the screen and uses the subroutine at line 620 to wait for a key to be pressed.

Lines 310-320 places the message for the second routine on the screen. This routine will convert a decimal number to binary.

Line 330 clears A$. A$ will be used again to store the conversion. The number to be converted will be stored in the DGIT variable.

Line 340 checks to see if a zero was entered. If it was, the computer will return to the main menu.

Line 350 checks to see if the number entered exceeds the highest possible entry. If it does, the computer will go back to line 330 to wait for another entry.

Line 360 stores the number entered in STDGIT.

Line 370 begins the conversion routine. V is set to one move that the largest number that could be entered. This value will be divided by two and used in the subroutine that sets the digits in the binary number. The binary number will be placed in A$.

Line 380 places the number and the binary number on the screen. It uses the subroutine in line 620 to wait for a key to be pressed. The computer will go back to line 310 to wait for another entry.

Lines 390-400 place the message for the third routine on the screen. This routine will convert a HEX number to decimal.

Line 410 sets the DGIT variable to zero. This variable will be used to store the decimal number. The VALIDATE options assures that only numbers and the letters A through F will be accepted. The HEX number will be stored in A$.

Line 420 checks to see if A$ is a null string. If it is null, the computer will be sent back to the main menu.

Line 430 finds out how many positions are in this number. The maximum position is four, but the number could contain one, two, or three as well.

Line 440 begins a FOR . . . NEXT loop to convert the HEX number to decimal. The length of the loop depends on the value of HEXP. The ASCII value of each position of A$ is placed in the C variable.

Line 450 subtracts 55 from C. If the value of C is less than 10, the character is a number and the computer takes the value of that position. If it is 10 or more, it has the value of that letter.

Line 460 calculates the decimal equivalent of the value of C based on the position of C in A$. Each position of the HEX string is a power of 16. By subtracting the value of V from HEXP, we know what power to raise 16 to. This number is then multiplied by the value of C and added to the contents of DGIT. Each time the computer travels through this loop, the power that 16 will be raised to is one less than the previous power, or, one position to the right.

Line 470 places the HEX number and the decimal equivalent on the screen. The computer then waits for a key to be pressed before going back to line 390.

Lines 480-490 place the message for the fourth routine on the screen. This routine will convert a binary number to decimal.

Line 500 clears the DGIT variable and waits for an entry. The VALIDATE option ensures that only ones and zeroes can be entered. The entry will be placed in A$.

Line 510 checks A$ for a null string. If it is null, the computer will return to the main menu.

Line 520 places the length of A$ into the P variable. Again, this variable will determine how many times the computer loops through the conversion routine.

Line 530 begins the conversion routine. The value of A$ at the position determined by P is stored in C.

Line 540 checks to see if this position is a one. If it is, the value of that position is added to the contents of DGIT. The position value is determined by subtracting the position that V is pointing to

from the number of positions in the string; then raising two to the power equal to that difference. This number is added to DGIT. In this routine we raise two to a power because each place value of a binary number is a power of two.

Line 550 continues the loop.

Line 560 places the decimal equivalent of the binary number on the screen and waits for a key to be pressed before going to line 480.

Line 570 is the first line of the subroutine used by the decimal to hex and decimal to binary routines. If the value of STDGIT is less than the value of V, there is no number for this position and zero is added to the contents of A$. The computer returns to the routine that sent it.

Line 580 finds the integer of the number that will be converted. The value of the position is divided into the number.

Line 590 multiplies the integer by the place value and subtracts it from the number that it is converting. This removes the place value from the number. The remainder will be converted for the next position.

Line 600 checks to see if the number is greater than nine. If it is, the number must be converted to a letter and placed in A$.

Line 610 places the number in A$.

Line 620 is the subroutine that the computer uses to see if a key has been pressed. It gives you time to copy the number and the conversion from the screen.

Index

The Last Word on the TI-99/4A

If you are intrigued with the possibilities of the programs included in *The Last Word on the TI 99/4A* (TAB BOOK No. 1745), you should definitely consider having the ready-to-run tape containing the software applications. This software is guaranteed free of manufacturer's defects. (If you have any problems, return the tape or disk within 30 days, and we'll send you a new one.) Not only will you save the time and effort of typing the programs, the tape eliminates the possibility of errors that can prevent the programs from functioning. Interested?

The programs are available on tape for TI-99/4A with 16K and Extended BASIC

at $19.95
for each tape or disk plus $1.00 each shipping and handling.

I'm interested in the program from *The Last Word on the TI-99/4A*. Send me:

_____ tape(s) for TI-99/4A with 16K and Extended BASIC (6510S)

_____ TAB BOOKS catalog

_____ Check/Money Order enclosed for $ _____

plus $1.00 shipping and handling for each tape ordered.

_____ VISA _____ MasterCard

Account No. _____ Expires _____

Name _____

Address _____

City _____ State _____ Zip _____

Signature _____

Mail To: **TAB BOOKS INC.**
P.O. Box 40
Blue Ridge Summit, PA 17214

(Pa. add 6% sales tax. Orders outside U.S. must be prepaid with international money orders in U.S. dollars.)

TAB 1745